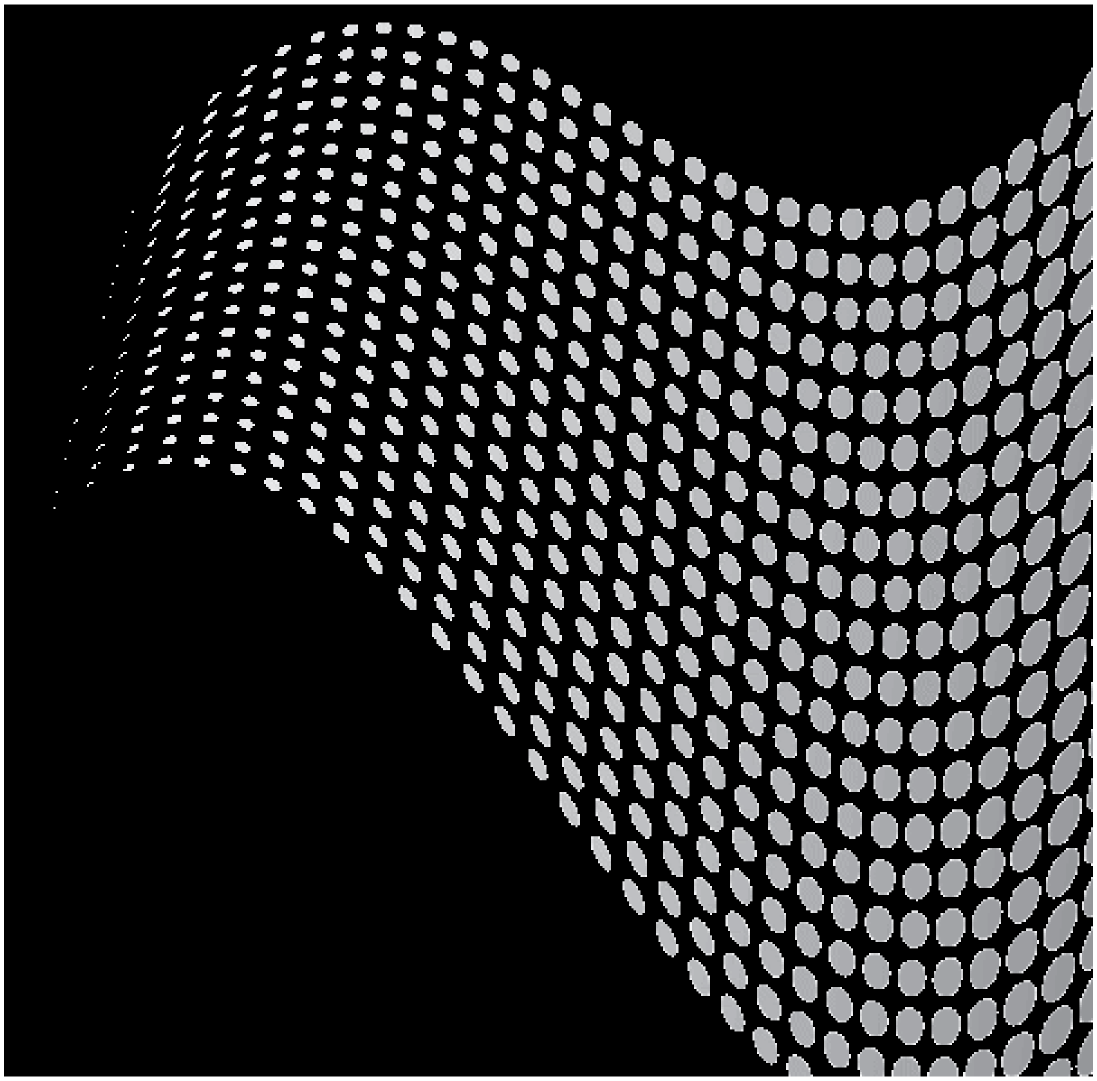
FAST Enterprise Search Platform

version:5.1

Advanced Linguistics Guide



Document Number: ESP1036, Document Revision: A, May 22nd, 2007

Copyright

Trademarks

Restricted Rights Legend

Contact Us

Web Site

Please visit us at: *http://www.fastsearch.com/*

Contacting FAST Corporate Offices

US Headquarters, Boston, MA
FAST
Cutler Lake Corporate Center
117 Kendrick Street, Suite 100
Needham, MA 02492 USA
Tel: +1 (781) 304-2400 (8:30am - 5:30pm EST)
Fax: +1 (781) 304-2410

Corporate Headquarters, Oslo, Norway
FAST
Torggata 2-4-6
N-0181 Oslo, Norway
Tel: +47 2301 1200
Fax: +47 2301 1201

Technical Product Support

Technical Product Support is offered to FAST subscribers with active FAST Maintenance and Support agreements. Please submit tickets and requests by using the Web-based ticketing system at *https://ticket.fast.no* or email *tech-support@fastsearch.com.*

If you do not have access to FAST ticket system, please contact Customer Relations at

Email: *customerservice@fastsearch.com*

For additional information such as phone numbers and times, please refer to your FAST Customer agreement.

Product Software and License

To request FAST licenses or software for customers, contact your FAST Account Manager or email *customerservice@fastsearch.com*

To request FAST licenses or software for partners, contact your Channel Sales Representative or email *partners@fastsearch.com*

For product evaluations, contact your FAST Sales Representative, FAST Sales Engineer, or Channel Sales Representative.

Product Training

E-mail: *fastuniversity@fastsearch.com*

Sales Inquiries

E-mail: *sales@fastsearch.com* or contact your FAST account manager.

Partner Inquiries

E-mail: *partners@fastsearch.com* or contact your Channel Sales Representative.

Obtaining Updates in Product Documentation for FAST ESP

Customer and Partner Extranet

You can check the customer and partner extranet for updated versions of product documentation. To obtain access to the extranet:

Send an email to this address *register-extranet@fastsearch.com* and include the following information:

- company
- email
- first and last name

Allow 48 hours during normal business hours for processing.

Note: Only companies with active Maintenance and Support (M&S) or active Partner agreement with signed Extranet Access forms are eligible. Standard M&S entitles a company to 3 users, and Premium 10 users. Partners, refer to your agreement for number of eligible users.

Contents

Chapter

1

Linguistics in FAST ESP

Topics:

Linguistics and Relevancy

In search linguistics is defined as the use of information about the structure and variation of languages so that users can more easily find relevant information. The document's relevancy with respect to a query is not necessarily decided on the basis of words common to both query and document, but rather the extent that its content satisfies the user's need for information.

Linguistics tools determine the intent behind keywords. For example, a user searching for *MP3* player would be interested in a hit that matched *iPod*. If the site only shows results for the keywords *MP3* and *Player*, a sale could be lost.

In order to achieve relevancy, linguistic processing is performed both at the document level – during document processing – and at the query level – during query and result processing. On the query side linguistic processing results in a query transformation, on the document side, linguistic processing results in document enrichment prior to indexing in order to cover grammatical forms and synonyms.

Linguistics is also used to extract relevant meta information out of unstructured text, turning unstructured content into structured content. This kind of information (like persons, companies, dates,prices) can for example be used for navigation in the results.

FAST ESP provides a comprehensive set of linguistic features.

Linguistic Concepts

The linguistic concepts offered by FAST ESP are explained in detail in this guide. Understanding the basic workflow and the different concepts makes it easier to understand how to best use linguistics.

Linguistics is used for different purposes:

- Basic linguistic processing: preparing the document for further processing and determining the language of the document.
- Normalization and approximate matching: allow simultaneous search for all variants, e.g. search across morphological or spelling variations, misspelled forms or phonetic variants.
- Conceptual relatedness: perform search across semantically related words, such as synonyms, acronyms or category denominators.
- Content extraction: extracting meta information from the document, typically meaningful entities. The extracted items can be used for narrowing down search results, allow navigation or performing search within the scope of one entity, for example a proper name.
- Filtering: blocking unwanted content

Basic Linguistic Processing

Basic linguistic processing is the prerequisite for all other linguistic operations and for indexing the document.

- Language and encoding detection determines the language and encoding of the documents to be indexed.
- Tokenization splits the text up into the actual words that should be indexed and processed further.

Normalization and Approximate Matching

Many of the standard linguistic modules cope with the problem of normalization. Being able to identify and properly treat variations in natural language increases the relevancy of search significantly. Variation occurs on different levels:

- The character level: an example here is the inconsistent use of accented characters in French. Mapping for example accented characters to not accented characters increases recall significantly. This process is called **character normalization** . Character normalization is applied on query and on document side.

- The spelling level: query terms can be misspelled or can use a uncommon spelling of a term. **Spell checking and spelling correction** helps to find the right documents even for misspelled queries. **Spelling variants** are correct spelling variations of the same word. They can be handled using the same engine as synonyms.
- The pronounciation level: typically names, especially foreign names, are often misspelled. **Phonetic search** uses phonetic rules to map words that are spelled differently, but sound the same, to a common phonetic representation. This is done on query and on document side.
- The morphological level: in many languages, a word can appear in different forms, for example the singular and plural form (such as *products* and *product*). **Lemmatization** is the aggregation of different word forms to enable search across different forms of the same word. It enables searches to match documents with similar meaning, but different word forms in the document or the query. Lemmatization has similar effects as stemming, but is more precise, as it based on dictionaries.
- The sentence level: parts of queries and query terms might not be relevant for searching. The **antiphrasing** mechanism allows to remove phrases like *where can I find information on* from queries. It can also be configured to remove simple **stop words** .

Most of the normalization is done on the query side and document side in the same way. Lemmatization and synonym expansion can be configured to be applied only to the document or only to the query.

Conceptual relatedness

- The semantic level: often, the same concept can be expressed by using different words, like for example *car* and *automobile* . Different words with the same or closely related meaning are called **synonyms** . **Synonym expansion** expands query or document terms with their synonyms in order to retrieve all documents that are about the same concept.

Content Extraction

There are several document processors dedicated to information extraction in order to improve the search experience by for example providing input to navigators. We distinguish between **entity extraction** , **noun phrase extraction** and **structural analysis (STAN)** .

- **Entity extraction** is isolating known linguistic constructs, such as proper names or location designators.
- **Structural Analyis** extracts complex items, such as addresses or the main text block from documents, making use of structural information (such as HTML-markup) in the document.
- **Noun phrase extraction** determines noun phrases based in linguistics algorithms. Complex noun phrases are the most relevant content descriptors for most text types and can be used for classification and drill down.

Filtering

The **Offensive Content Filter** is a document analysis tool to filter content regarded as offensive.

The filter is implemented as a separate document processor that can be added to an ESP pipeline.

General Linguistics Settings in ESP

Installation Options

The following installation options of ESP directly influence linguistics processing:

- *Select default language for advanced linguistics processing of queries*: Allows choosing the default language for spell checking and lemmatization (queries only). Only the pre-installed languages shipped with the product are available.

- *Select languages for advanced linguistics processing*: Here you can choose the languages for which you wish to activate advanced linguistics (spell checking and lemmatization). Multiple languages can be selected here.
- *Choose index-profile type*: Here you can choose, whether you wish to install an index profile with lemmas (*web profile with lemmas*). If you want to enable lemmatization in your system, you should choose a profile with lemmas.

Both the default languages and the active languages can also be changed after installation language can for spell checking and lemmatization, independently from each other. You can only select languages shipped with the product during installation.

Note: Advanced Linguistics support for languages not shipped with the product can be added after the initial installation. Contact your FAST Account Manager or FAST Technical Support for details.

Note: Configuration of lemmatization of Korean and Japanese is not exposed in the installation menu. If you install an index profile with lemmas, lemmatization support for those language is added automatically.

Changing the Default Query String Language

This topic describes how to change the default language in which a query string is expected to be written in FAST ESP by the `didyoumean` framework and the `lemmatizer` query processor.

The query parameter `LANGUAGE` can be used to set the language of the query string in order to apply language specific query processing such as spell checking and lemmatization. If this parameter is not set, the lemmatizer stage on query side and the spell checking framework use the default language specified during installation. You can change this default language for both stages independently.

If you want to define another default language for spell checking or lemmatization (which applies when the indicated query parameter is not used), this can be done by setting the appropriate parameters for the query processors.

1. To change the default language applied to spell checking, open `$FASTSEARCH/etc/config_data/QRServer/webcluster/etc/qrserver/qtf-config.xml`
2. Change the value of the `default.language` parameter in instance didyoumean to the desired language code.
3. Refresh search views. `$FASTSEARCH/bin/view-admin -m refresh`

   In a multinode setup, this command must be performed on all Query and Result servers.
4. To change the default language applied to lemmatization, open `$FASTSEARCH/etc/LemmatizationConfig.xml`
5. In the `lemmatization` tag, change the attribute `default_query_language` to the desired language code.

   In a multinode setup, this command must be performed on all Query and Result servers.

Linguistics at Runtime

Languages during Document Processing

The document's language is auto-detected in most pipelines and is stored in the `languages` attribute using the ISO-639-1 language code. The primary language of each document is stored in the `language` attribute. Refer to the chapter on language and encoding detection (*Language and Encoding Detection* on page 17) for a list of supported language codes.

During document processing, the behaviour of the linguistic processors depends on the language of the document. The language influences the behaviour of:

- tokenization
- lemmatization

- document side synonyms
- entity exctraction
- noun phrase extraction
- structural analysis (STAN)
- vectorization
- phonetic normalization

Languages during Query Processing

The language of queries has to be set manually for each query and cannot be auto-detected. Use the query parameter `language` for this purpose (the language can also be set in the Search Front End with a pull-down menu).

Note: The `language` **query parameter** is independent from querying both the `language` or the `languages` field in the query itself. The query parameter serves to select dictionaries and procedures, whereas a query for the information in the data fields just selects documents with a specific language, without setting the query parameter.

The following query features use make use of the language tag:

- spell checking
- tokenization
- lemmatization
- phonetic normalization

If no query languages is set, lemmatization and spell checking use the configuration for the default language defined for each stage. The tokenization service uses the default tokenizer.

Linguistics Features During Query Processing

The behaviour of the different linguistics query transformers can be configured on a per query basis with a couple of parameters. Refer to the chapters about the different features for more information.

The parameter `linguistics=off` can be used to turn off the linguistics submodules spell checking and lemmatization for a particular query string, e.g. `string("wilder",linguistics=off)` will prevent the query string from being lemmatized and spell checked.

Handling Language Codes

Languages with multiple language codes

Some languages have multiple language codes and therefore special processing considerations may apply.

Chinese

ISO-639-1 does not have different codes for the writing systems of traditional and simplified Chinese. However, for a search engine it is important to distinguish between the two variants. For this purpose, the languages codes *zh-simplified* and *zh-traditional* are used during language detection and should also be used:

- when you indicate the query parameter `language` on the query side
- you will search for documents in one of the two Chinese writing systems in the `languages` field.

When you specify *zh* only in the query parameter `language` , both traditional Chinese and simplified Chinese fields are retrieved.

The ISO-639 language codes with country suffixes, *zh-tw, zh-hk* and *zh-cn, zh-sg* will be mapped to zh-traditional and zh-simplified if they occur in the documents meta data.

Norwegian

There are two official written Norwegian languages, Bokmaal and Nynorsk.

The language detection document processor is able to detect the two variants of Norwegian (*nb* for Bokmaal and *nn* for Nynorsk). The lemmatizer and the spell checking modules do not use different modes for both Norwegian languages, but use combined dictionaries. To assure correct processing, set the query parameter `language` to Norwegian for all Norwegian queries (Nynorsk and Bokmaal).

To retrieve documents in Norwegian by querying the `languages` field, you have to use a conjunction of both specific language codes (*nn* , *nb*). This can be done as follows in FQL:

```
AND(FILTER(languages:OR("nn","nb")),your ordinary query expressions)
```

Country Suffixes and Country-Specific Language Variants

In the document meta information for XML or HTML, ISO 639 language codes can be supplemented by a country variant indication, such as *en-us* , *en-uk* or *en-au* for English in its US-American, British or Australian variant. Detection of variants, and different processing of language variants with different country suffixes is not supported in standard installations. This applies to both document processing and query processing. It is possible to configure a custom setup to support language variants, and synonym dictionaries to map variants that are available. Contact your FAST account manager to get dictionaries for the English variants.

Also for some other languages (e.g. French), variant-specific dictionaries and variant synonyms are available.

Language Specific Solutions

Natural languages may have specific properties which require special solutions. This section gives an overview over language-specific challenges for search engines in general and describes solutions in ESP.

Note: Properties and processors for Asian languages (specifically Chinese, Japanese, Korean and Thai) are discussed in a dedicated separate chapter.

Arabic

Arabic is written from right to left, but no special processors are needed for Arabic to handle this property, as all encodings use the correct byte order, i.e. no so called *visual* encoding exists for this language.

Arabic has optional vowel diacritics, which are not used in normal text. For consistent search, those vowel diacritics should be removed in the tokenization process. The same is true for other languages written in Arabic script (Farsi, Urdu, Pashto). Contact your FAST Account Manager or FAST Technical Support if you need help with setting up a normalization configuration.

Arabic has a rich morphology, which is handled by FAST's lemmatizer modules and the Arabic advanced linguistics package.

Finnish

Finnish has an extremely rich morphology. If lemmatization is set up for this language, lemmatization by reduction is the only viable strategy.

Hebrew

Hebrew is written from right to left. Some documents in Hebrew are encoded in ISO-8859-8 visual encoding. In this encoding, the byte stream contains the characters in reverse order. The byte stream in those documents has to be reversed. For this purpose the `HebrewVisualToLogical` processor can be integrated in the document processing pipeline before the `Tokenizer`. It will automatically reverse the inverted text.

Hindi

Indian languages are written in Indic scripts. The most widespread script for writing Hindi is *Devanagari*. Unfortunatly, since historically no widely accepted character encoding for this script was available, many web pages in Hindi use a custom font specification instead. Such pages cannot be converted into Unicode. Only documents encoded in ISCII encoding or UTF-8 can be processed in the search engine.

An advanced linguistics package is available for Hindi lemmatization, spell checking and anti-phrasing.

Norwegian

Norway has two languages, Bokmaal and Nynorsk. Refer to *Handling Language Codes* on page 15 for details.

Serbian

Serbian is commonly written in Cyrillic script. Serbian documents in Latin script will not be identified as such, but will be identified as Croatian.

Language and Encoding Detection

During document processing, documents can be analyzed to detect the language and encoding in which they are written. This functionality is provided by the Automatic Language Detection feature.

Detecting the language of a document is essential to all other linguistic analysis features, as the resulting language information is used to select language-specific dictionaries and algorithms during document processing and query processing.

During language detection, a given document is analyzed for all supported languages. For each language, a certain score is calculated, based on the occurrences, number, and length of certain test strings. The language(s) that reaches the highest score, and for which the score exceeds a preset threshold, are specified as the document languages.

Attention: For queries, the language has to be explicitly set by the end-user or search application, as the query itself generally provides too little context for determining the language it is written in.

Supported Languages for Automatic Language Detection

FAST ESP automatically recognizes 81 different languages in all common encodings.

This is a list of all the languages that can be automatically detected by the `LanguageAndEncodingDetector` processor during document processing. In ESP, languages are stored using the ISO-639-1 two-letter languages codes.

Language	ISO 639 Code	Language	ISO 639 Code
AFRIKAANS	af	JAPANESE	ja
ALBANIAN	sq	KAZAKH	kk
ARABIC	ar	KIRGHIZ	ky
ARMENIAN	hy	KOREAN	ko
AZERI	az	KURDISH	ku
BANGLA	bn	LATIN	la
BASQUE	eu	LATVIAN	lv
BOSNIAN	bs	LETZEBURGESCH	lb

Language	**ISO 639 Code**	**Language**	**ISO 639 Code**
BRETON	br	LITHUANIAN	lt
BULGARIAN	bg	MACEDONIAN	mk
BYELORUSSIAN	be	MALAY	ms
CATALAN	ca	MALTESE	mt
CHINESE (SIMPLIFIED)	zh-simplified	MAORI	mi
CHINESE (TRADITIONAL)	zh-traditional	MONGOLIAN	mn
CROATIAN	hr	NORWEGIAN (BOKMAAL)	nb
CZECH	cs	NORWEGIAN (NYNORSK)	nn
DANISH	da	PASHTO	ps
DUTCH	nl	POLISH	pl
ENGLISH	en	PORTUGUESE	pt
ESPERANTO	eo	RHAETO-ROMANCE	rm
ESTONIAN	et	ROMANIAN	ro
FAEROESE	fo	RUSSIAN	ru
FARSI	fa	SAMI (NORTHERN)	se
FILIPINO	tl	SERBIAN	sr
FINNISH	fi	SLOVAK	sk
FRENCH	fr	SLOVENIAN	sl
FRISIAN	fy	SORBIAN	wen[1]
GALICIAN	gl	SPANISH	es
GEORGIAN	ka	SWAHILI	sw
GERMAN	de	SWEDISH	sv
GREEK	el	TAMIL	ta
GREENLANDIC	kl	THAI	th
HAUSA	ha	TURKISH	tr
HEBREW	he	UKRAINIAN	uk
HINDI	hi	URDU	ur
HUNGARIAN	hu	UZBEK	uz
ICELANDIC	is	VIETNAMESE	vi
INDONESIAN	id	WELSH	cy
IRISH (GAELIC)	ga	YIDDISH	yi
ITALIAN	it	ZULU	zu

Language and Encoding Detection and Encoding Conversion

The text language and encoding can either be defined explicitly in the document's metadata, or determined by an automatic process during document processing. FAST ESP uses this information to select the appropriate language-specific dictionaries and algorithms during document processing.

[1] No two letter code is available for this language.

The standard document processing stage for language and encoding detection is the `LanguageAndEncodingDetector`. In addition, the `LanguageDetectorTextParts` can be integrated in a document processing pipeline to assign a language to each text section in case of multilingual documents. The standard processor for encoding conversion to utf-8 is the `EncodingNormalizer`.

Language Detection

The language of each document is automatically detected. For each of the supported languages, a score is calculated. If no language passes the acceptance score, a fallback value is used. In case of multilingual documents, a maximum of two languages will be assigned by the `LanguageAndEncodingDetector` The `LanguageDetectorTextParts` can assign a language to each text section, and will also set the attribute 'languages' to all languages it has detected. In case the language is specified in the document's metadata, the outcome of the automatic process is only used to sanity check the metadata, and a specific, very low acceptance score is used for the language specified by the meta tag. Running such plausibility checks against documents containing language metadata is necessary in order to detect documents where the meta information is misleading.

Default Language

If the language of the document cannot be determined, a value of "unknown" will be specified for the document element.

The fallback value can be set in the parameter FallbackLanguage in the LanguageAndEncodingDetector.

Encoding Detection and Conversion

During language detection, the encoding of the document is detected as well. Encoding detection is crucial to be able to normalize the content to Unicode (UTF-8). This is important because all processors within FAST ESP operate on Unicode encoded documents and queries. In case the encoding is specified in a document's meta information, the encoding is always accepted by the `LanguageAndEncodingDetector`, unless the processor is able to detect byte sequences that are invalid for the given meta encoding.

If the encoding of a document is not UTF-8, the document is converted to this encoding after the detection stage by the `EncodingConverter`

Dictionaries and Data Files

Language detection is based on dictionary and data files which cannot be edited.

Language and Encoding Document Processors

The following sections describe processors used for language and encoding detection and encoding conversion.

LanguageAndEncodingDetector
This document processing stage automatically detects text language(s) and encoding format (for non-scope fields) in a document.

Parameter	Type	Value	Description
ContentAttribute	str	html data	The field containing the document content. If comma-separated fields are specified, only the first field containing content will be processed.
URLAttribute	str	url	Document attribute that contains the url of the document. If this field is empty, the country domain (e.g. 'uk') is not used for language detection, and no domain suffix specific fallback language is set.
FallbackEncoding	str	iso-8859-1	The fallback encoding if no encoding can be detected automatically.

Parameter	Type	Value	Description
FallbackLanguage	str	unknown	The fallback language if no language is detected.

Description

Note: The encoding format in scope fields is detected by the `XMLParser` processor.

The order of preference is: explicit metadata, content, and URL. If any of the language or charset attributes are preset, the values are used to sanitize the detection.

The fields in ContentAttribute are a preference list; only the first field that has content is processed

The FallbackEncoding and FallbackLanguage values are used if none of the above strategies detects an encoding or language, respectively.

Usage

This processor is preconfigured for optimal use in pipelines with all document formats.

In special cases, such as structured data with limited amount of text, it may be hard to provide a reliable detection. For this purpose you can define a fallback setting for language and encoding.

FallBackEncoding should be set to a value that matches the features of the application. E.g. for Asian languages, the most predominant Asian encoding (e.g. EUC-JP or SJIS) of the application should be used.

For a deployment that is to operate primarily on Asian data, it is recommended to set the fallback encoding to the most likely one for that particular environment. In Japan for instance, this would often be *shift-jis* .

LanguageDetectorTextParts

This processor automatically detects and assigns a language to a section of text.

This processor assigns a language to each text section provided by the html-parser. Languages that have been assigned to the entire document in the earlier run of the `LanguageAndEncodingDetector` will receive a boost. This will enable language-specific tokenization and lemmatization on a per section basis. If this stage is not included, tokenization and lemmatization will be carried out based on the document language.

Tip: The processor must be placed after the HTML parser in the pipeline. This is because the HTML parser divides documents into sections and the LanguageDetectorTextParts processor operates on these sections.

Parameter	Description	Default value
overwrite_nodelang	If this parameter is set to '1', an already specified language for a section is replaced by the detected language.	1
attributes	A space separated list of fields to be searched.	title headings body
configfile	Defines the name of the configuration file for the language identifier.	etc/langid/config.txt

TextLanguageDetector

This document processing stage detects the language and encoding format of plain text attributes.

Parameter	Type	Value	Description
Attributes	str	data	Document attribute that contains the url of the document. If this field is empty, the country domain (e.g. 'uk') is not used for language detection, and no domain suffix specific fallback language is set.
FallbackLanguage	str	unknown	The fallback language if no language is detected.

Parameter	Type	Value	Description
FallbackEncoding	str	iso-8859-1	The fallback encoding if no encoding can be detected automatically. This should be set to a value that matches the features of the application. E.g. for Asian languages, the most predominant Asian encoding (e.g. EUC-JP or SJIS) of the application should be used.

Description

The processor output is in the attributes language , secondarylanguage , languages , and encoding . A common language and encoding is selected for all input attributes. The fallback parameters are used when the automatic detection fails, possibly because of too little or ambiguous input.

XMLLanguageDetector

This document processing stage detects the language of XML attributes.

The processor output is in the attributes `language` , `secondarylanguage` and `languages` . A common language is selected for all input attributes.

Parameter	Description	Default value
Attributes	The attributes are either a DOM tree or have "dom" metadata from XMLParse. Format: attributename[:xpath] where the content of the selected nodes is used for language detection.	xml (no xpath, all content of the DOM tree is used.)
FallbackLanguage	The fallback language if no language is detected.	unknown

EncodingNormalizer

This document processing stage converts content of HTML elements from one encoding format to another.

Parameter	Type	Value	Description
Encoding	str	charset	This parameter should specify the name of a document element that contains the desired output character set.
Input	str	html	Name of element that includes the content that should be converted.
Output	str	html	Name of element that should contain the content after conversion.
FallbackEncoding	str	iso-8859-1	Encoding to use if encoding specified in the Encoding parameter fails.
StrictUTF8Checking	int	1	If the input is declared as utf-8 and the input contains invalid UTF-8 characters, the input is considered as being FallbackEncoding if StrictUTF8Checking is 1. utf-8 is accepted and invalid characters are replaced by the generic replacement character if it is 0.

Usage

The `EncodingNormalizer` is used in most pipelines. It should always be placed after the document processing stage that is responsible for language and encoding detection.

UTF8Encoder

This document processing stage converts specified document elements into valid UTF-8 encoded elements.

Parameter	Type	Default value	Description
attributes	str		A space separated list of element pairs where each pair has the format `input:output` .

Parameter	Type	Default value	Description
encodingattribute	str		If the document specifies encoding in an attribute other than `encoding` , specify that attribute using this parameter.
encoding	str		If `encodingattribute` is not set, this parameters value is investigated.

Description

`input` is the source element (text string) and `output` is the destination element that will contain the format converted element. Existing content of the destination elements will be overwritten. Element that is not yet defined in the pipeline will be ignored.

Encoding format could potentially be set on each element in a document. When a document is processed FAST ESP looks for encoding metadata in one of the following sources in priority order. Input elements must be encoded in of the formats supported by the the Python module `pyiconv` . A list of the supported formats can be printed on the command line by executing the following command on the machine where FAST ESP is installed: `iconv -l`

- The encoding metadata attribute in the document element. This is for advanced use only, normally only used when coding your own document processors.
- Get encoding data from the element specified in the encodingattribute parameter.
- Use the fallback encoding in the encoding parameter.

Usage

This stage may be used if you are using one of the Database/XML pipelines and your content is not according to UTF-8. FAST ESP requires that elements are encoded using UTF-8.

Configuring Automatic Language Detection

Language and encoding detection for entire documents is part of most preconfigured document processing pipelines (e.g. `Generic`). If you want to add language detection to any custom pipeline, you have to integrate it into one of the document processors described in the previous sections. Generally, for a pipeline with mixed document formats, it is recommended to use the the `LanguageAndEncodingDetector`.

Language and encoding detection on **text sections** is only part of the `CustomLinguistics` pipeline. For pipelines based on any other template, the `LanguageDetectorTextParts` has to be added manually.

Applying Automatic Language Detection

This procedure describes how you should set up automatic detection of languages in documents during document processing.

1. Select **Document Processing** in the Admin GUI.
2. Click on the **plus** sign to the right for the correct pipeline.
3. Enter a name and description for the new pipeline.
4. Add the chosen language detection stage to the list of pipeline stages.
5. Click **submit**.

Applying Automatic Language Detection on Text Sections

This procedure describes how you should set up automatic detection of languages for individual text sections during document processing.

1. Select **Document Processing** in the Admin GUI.
2. Select the pipeline where you want to add the processor.

3. Add LanguageDetectorTextParts to the list of pipeline stages. It must be added after the HTML-parser.
4. Click **submit**.

Enable Fault Tolerant UTF-8 Processing

This task describes how to enable error tolerant UTF-8 document processing.

By default, ESP will not accept documents as UTF-8 if they contain byte sequences which are not valid for this encoding. If you want to enable UTF-8 processing which tolerates such sequences, you can configure the LanguageAndEncodingDetector to accept them, and the EncodingNormalizer to skip non-convertible bytes. Characters in character sequences which are not convertible will then be replaced by the Unicode replacement Character U+FFFD.

1. Open the file `$FASTSEARCH/resources/dictionaries/util/langid/encoding.txt` and locate the active section headed by `START utf-8 u` where valid sequences for UTF-8 encoding are defined. Locate the (commented out) section defining the error tolerant UTF-8 encoding detection below the active section.

   It is advisable to backup the original file before editing.
2. Comment out the active section by putting "//" before each line and uncomment the inactive section for UTF-8.

   Take care not to uncomment any textual explanations.
3. In the document processing interface, create a processor EncodingNormalizer1 based on the EncodingNormalizer.
4. Set the parameter StrictUTF8Checking to value '0' in the EncodingNormalizer1
5. Replace the EncodingNormalizer with EncodingNormalizer1 in all pipelines you are using.

   If you keep pipelines containing the default EncodingNormalizer, UTF-8 documents with invalid byte sequences will be dropped when processed through those pipelines.

Chapter 2

Tokenization and Normalization

Topics:

For most languages, such as English, this is a rather trivial operation. Characters such as spaces, tabs, periods, commas, dashes, question marks, and quotes are examples of word delimiters. In the following, when we refer to tokenization we imply tokenization and normalization.

- Tokenization is performed, both during document and query processing, on any content that is configured as `tokenized="auto"` in the index profile.
- Dictionaries that are applied to tokenized content, like lemmatization and synonym dictionaries, are tokenized and normalized in the same way as indexed content and queries. Dictionaries that are compiled using the dictionary managment tools are automatically tokenized correctly; dictionaries shipped with the product are pre-tokenized based on the pre-installed tokenizer configuration.
- Tokenization can be language specific, but there is always a default configuration, covering *all other languages*, the so called default tokenizer (*The Default Tokenizer* on page 30).

For information on language specific tokenization, see the *Tokenization for Asian Languages* on page 102 .

Architecture

The FAST ESP tokenization service consists of the following modules

- A configurable, language independent tokenization engine, the so called generic tokenizer (see *Generic Tokenizer* on page 28).
- Additional language specific tokenizers can be added through tokenizer plug-ins (see *Add a Language Specific Tokenizer* on page 32), that can also annotate morphologic and semantic information.
- Normalization of tokens

Tokenization Process

Before tokenization, the input is normalized based on the input normalization settings. The tokenization process proper then consists of three steps:

1. Optional, language dependent tokenization and annotation
2. Tokenization by the default tokenizer.
3. Normalization of tokens into their indexed form.

The first step, the language specific tokenization, is only executed if there is a specific tokenizer for this language. Most European languages are handled by the second and third step only.

	Default Tokenization	**Language Specific Tokenization**
Input	An example, again.	検索エンジン Search Engine はインターネットを検索する。
1.		検索エンジン/(/Search/ /Engine/)/は/インターネット/を/検索/する/。
2.	An/example/again	検索エンジン/Search/Engine/は/インターネット/を/検索/する
3.	an/example/again	検索エンジン/search/engine/は/インターネット/を/検索/する

Customizing the Tokenization Process

This topic describes the basic elements of the tokenization process and what you need to consider when you want to customize the process.

In FAST ESP, tokenization incorporates an optional language-specific morphological analysis, including segmentation and annotation, a language-independent default tokenization and finally the normalization of the resulting tokens. Additionally, all content is normalized by the so-called *Input Normalization* on page 35 prior to tokenization.

The behavior of all of the above-mentioned elements of the tokenization service is configured by a single tokenization configuration file. The purpose and the syntax of this file is described in the remainder of this chapter. The default location of the tokenization configuration is at `$FASTSEARCH/etc/tokenizer/tokenization.xml` (`%FASTSEARCH%\etc\tokenizer\tokenization.xml on Windows).` and has to be be identical on all nodes of an installation. It is best not to manually deploy this file, but instead to use the *Deploying a Changed Tokenization Configuration* on page 27

Attention:

Making a change in the tokenization configuration effectively requires you to:

- Delete all collections.
- Deploy the new configuration.
- Refeed all content in the installation.
- If you have lemmatization installed on your system, you need to recompile the lemmatization dictionaries (especially if the tokenizer change affects the set of word characters or involved character normalization).

The tokenization configuration can only by changed in a new installation. It must remain unchanged as soon as there is content indexed, or else the tokenization of the queries and the indexed content will be inconsistent.

Deploying a Changed Tokenization Configuration

After the tokenization configuration has been changed, it needs to be deployed to all nodes, certain dictionaries need to be recompiled and the document and query processing has to be reset. The `config-deploy` tool should be used to do that.

After changing the tokenization configuration, these changes need to be transferred to all nodes. Additionally, dictionaries such as for lemmatization need to be recompiled applying the new tokenization rules. Afterwards, these dictionaries, too, have to be transferred to all system nodes. `config-deploy` handles the details of these operations transparently. It is also possible to publish the tokenization configuration, but not activate the changes at the same time, but for instance during the maintenance time window. `config-deploy -h` will output more detailled information about the options of this tool.

1. Make a copy of your currently used configuration, `$FASTSEARCH/etc/tokenizer/tokenization.xml` .
2. Set, publish and activate the new configuration file using `config-deploy` .
   This command will publish the tokenization configuration contained in the file `mynewconfig.xml` to all nodes and make the document processing a query processing subsystems activate the changes.
   ```
   config-deploy -f mynewconfig.xml -p -a
   ```

Tokenizers

Every tokenizer consists of a tokenizing instance and one or more normalizers that are applied to the resulting tokens. There are two different kinds of tokenizing instances as well as two different types of tokenizers. The available tokenizing instances are

- The built in generic tokenizer that can performs character based tokenization and is best suited for most European languages.
- Tokenizer plugins (*Tokenizer Plug-Ins* on page 31) provide the possibility of extending the tokenization service by tokenizing entities for specific languages and purposes (such as text segmentation for CJK languages).

There are two different types of tokenizer elements.

- The *The Default Tokenizer* on page 30 is applied to all content in fields that are configured as `tokenized="auto"` in the index profile. It is language independent and always a generic tokenizer. The generic tokenizer of the default tokenizer is also applied to the output of every language specific tokenizer. It is the ultimate instance defining the word and non-word characters of the tokenization service.
- A language specific tokenizer is only applied to content in the languages it is assigned to. Its tokenizing instance can be either a generic tokenizer or, more typically, a tokenization plug-in. In contrast to the default tokenizer, a language specific tokenizer specifies semantically interesting tokens. It has not to deal with the word character distinction, since its output is tokenized again by the default tokenizer.

Note: A tokenizing instance, like a generic tokenizer or a plug-in can be used in multiple tokenizers, which helps conserving memory.

Generic Tokenizer

The Generic Tokenizer performs tokenization of text in languages with word-boundary marks, like typical European texts.

The primary function of the generic tokenizer is to distinguish word from non-word characters. It aggregates continuous strings of word characters as tokens in a greedy fashion, i.e. it makes tokens as long as possible. Effectivly each token starts after a non-word character and ends right before the next non-word character in the text.

The behavior of the generic tokenizer can be controlled by specifying word and non-word characters.

Character Sets

Like most components and services in FAST, ESP the tokenization service works on Unicode characters. A *character set* of the generic tokenizer is a subset of all Unicode characters.

In the tokenization configuration, a character set is defined by a series of single code points, like

```
<character value="0x3F" />          <!-- '?' question mark -->
```

and continuous ranges (with boundaries included) of code points.

```
<range from="0x30" to="0x39" />     <!-- '0'-'9' numerals -->
```

As shown in the examples, characters are given as hexadecimal numbers.

For example, let's define the set of characters that are needed to write hexadecimal numbers in the way they are expected in the tokenization configuration:

```
<set name="hexadecimal">
 <range from="0x30" to="0x39" />     <!-- '0'-'9' -->
 <range from="0x41" to="0x46" />     <!-- 'A'-'F' -->
 <character value="0x78" />          <!-- 'x' -->
</set>
```

A character set is defined as an element `set` with a mandatory attribute `name` that has to be unique within the tokenization configuration file. A `set` contains a number of `character` and `range` sub-elements. Every `set` has to be a child element of `//configuration/charactersets`.

Character Set Types

The generic tokenizer distinguishes between three sets of characters:

- Non-word characters, and , that is the set of characters that are not part of a token. Members of this set are called **SKIP** characters and the set itself is denoted *skip set*.
- The set of greedy matched word characters, the so called **KEEP** characters. Usually, KEEP characters constitute and . A consecutive string of KEEP characters is emitted as one token, starting right after the preceding non KEEP character and ending right before the next non KEEP character.
- The **SPLIT** character set contains word characters that are emitted as individual tokens. Whenever the tokenized text contains a SPLIT character, this character will become a token that contains this single character only.

The three classes of character sets are mutually exclusive: a character is never in more than one set. Furthermore, the sum of all three sets has to cover the full Unicode range.

Since character sets are mutual exclusive, it is generally sufficient to define the SKIP and SPLIT characters. The KEEP characters are all remaining characters that are neither in the SPLIT nor in the SKIP character set.

Note: The implicit KEEP set automatically solves the requirement of coverage of whole Unicode range.

Specification of a Generic Tokenizer

A generic tokenizer is specified as a `generic-tokenizer` element in `/configuration/tokenizers`. It has a mandatory attribute `id` that has to be unique.

```
<generic-tokenizer id="my_tokenizer" >
 <skip  name="My-non-word-characters" />
 <split name="My-SPLIT" />
</generic-tokenizer>
```

The generic tokenizer contains an obligatory element `skip`. The `name` attribute of this element references the `name` of a character set. This character set specifies the SKIP characters of this generic tokenizer.

The optional element `split` defines a SPLIT set accordingly. Note that both sets may not overlap and that all characters that are in neither are implicitly word characters (ie, are in the KEEP set).

Note: Typically the SPLIT character set contains punctuation characters, if such characters are to be indexed and searched. Since this is generally not the case, typical generic tokenizers have no or a very small SPLIT character set.

Add a Generic Tokenizer

This topic describes how to add a generic tokenizer to the tokenization configuration.

The behaviour of a *Generic Tokenizer* on page 28 is controlled by a number of character sets.

Note: Due to system requirements the codepoints 0x00 to 0x20 and 0x7F - 0xA0 (control and space characters), must always be SKIP characters and never be indexed or part a tokenized query. It is further recommended to leave all characters defined as space characters in Unicode as SKIP characters.

1. Make a copy of your currently used configuration, `$FASTSEARCH/etc/tokenizer/tokenization.xml` .
2. Declare a `generic-tokenizer` element within the `tokenizers` element and include the mandatory unique id.

   ```
   <generic-tokenizer id="my_tokenizer"></generic-tokenizer>
   ```

3. Specify a `skip` element within the `generic-tokenizer` that defines the SKIP character set of your generic tokenizer.

   ```
   <generic-tokenizer id="my_tokenizer">
     <skip name="my_skip_set" />
   </generic-tokenizer>
   ```

   Note that the `skip` element is mandatory set must reference an existing character set.
4. Optional: Add a `SPLIT` set.

   ```
   <generic-tokenizer id="my_tokenizer">
     <skip name="my_skip_set" />
     <split name="my_split_set" />
   </generic-tokenizer>
   ```

   my_split_set has to be an existing name of a character set and can not contain any code points that is also in *my_skip_set* . All characters that are neither in *my_split_set* or in *my_skip_set* are implicitly word-characters.
5. Follow the *Customizing the Tokenization Process* on page 27 to deploy the new tokenization configuration.

The Default Tokenizer

The *default tokenizer* defines the tokenization behavior of Fast ESP with respect to word and non-word characters for all content of fields that are defined as `tokenize="auto"` in the index profile.

The default tokenizer is language independent and always a generic tokenizer. Its main purpose is to distinguish between word and non-word characters. The generic tokenizer defined as the default tokenizer is also applied to the output of every language-specific tokenizer. It is the single instance where word and non-word characters are defined in the tokenization service.

A default tokenizer is specified by a `default-tokenizer` element in `/configuration/tokenizers`. There is one and only one default tokenizer for each tokenization mode. Like every tokenizer, the default tokenizer element contains a mandatory tokenizing instance, which has to be a generic tokenizer. It therefore always contains a `generic` sub element with identifies a valid generic tokenizer instance by its `name` attribute:

```

<default-tokenizer>
  <generic name="my_tokenizer" />
  <normalization ... />
</default-tokenizer>

```

Additionally the default tokenizer contains the usual normalization specifiers.

Note: The normalization specification of a default tokenizer is only applied to content that is *not* primarily tokenized by a language specific tokenizer. Otherwise, the normalizers of this primary tokenizer are applied and only the generic tokenizing instance of the default tokenizer is used for the secondary tokenization.

Tokenizer Plug-Ins

Tokenization plug-ins allow the extension of the tokenization service by alternative tokenizing instances. Every language specific tokenization instance is implemented as a plug-in that can be deployed on demand.

The declaration of a tokenization plug-in consists of an element *tokenizer-plug-in* within *tokenizers*. The *tokenizer-plug-in* element must have an unique *id* attribute. Further it requires a plug-in specific *config* sub-element and the two mandatory attributes *library* and *set-up* that should be described in the documentation coming with the plug-in

The following is an example for the declaration of a tokenization plug-in.

```
<tokenizer-plug-in id="Japanese_Plug_in" library="basis-jma-plugin" set-up="pooling" >
 <config>
  <basis-jma>
   <compound-analysis />
   <dictionary-forms />
   <lemmatization add-reading="true" />
  </basis-jma>
 </config>
</tokenizer-plug-in>
```

Note that the only value in a plug-in declaration you usually are free to choose is the id of the resulting tokenization instance. All other values have to be set according to the documentation of the plug-in

Language Specific Tokenizer

A *language specific tokenizer* is used for the primary segmentation of text in the specific language.

A language specific tokenizer is only applied to content in the languages it is assigned to. Its tokenizing instance can be either a generic tokenizer or, more typically, a tokenization plug-in. In contrast to the default tokenizer, a language specific tokenizer emits rather morphologically correct tokens. It does not deal with the distinction between word and non-word characters, since its output is always subject to a secondary tokenization by the default tokenizer.

A language specific tokenizer is defined as a tokenizer element in /configuration/tokenizers. It has a mandatory attribute language with a list of all languages this tokenizer should be used for. The language definition has to be a space separated list of lowercase ISO-639-1 codes, such as assigned by the language detector (see *Supported Languages for Automatic Language Detection* on page 17). Like every tokenizer element, the tokenizer contains a mandatory tokenizing instance, which could be a generic tokenizer. More typically it will be a tokenization plug-in that is specified using a plug-in sub element that identifies a valid plug-in instance by its name attribute:

```
<tokenizer language="ja">
  <plug-in name="Japanese_Plug_in"/>
  <normalization transformation="lowercase" canonical="true" />
</tokenizer>
```

Additionally the tokenizer contains the usual normalization specifiers, which are typically, but not necessarily, identical to the ones of the default tokenizer.

Note: Document-side language specific tokenization depends on correct language detection and query-side tokenization requires the corect selection of the query language.

Add a Language Specific Tokenizer

This topics describes how to add an additional language-specific tokenizer to the tokenization process.

Assume your plug-in file is installed in `$FASTSEARCH/lib/libmy_plugin.so` (or `%FASTSEARCH%\lib\my_plugin.dll` on Windows).

The tokenization service is extensible by loading *Tokenizer Plug-Ins* on page 31 . In order to install such a plug-in, you have to deploy it on all nodes of your installation in accordance to the accompanied documentation. In general such a plug-in consists of a single plug-in library file and some additional data.

1. Make a copy of your currently used configuration, `$FASTSEARCH/etc/tokenizer/tokenization.xml` .
2. Add a `tokenizer-plug-in` element within the `tokenizers` element and include the mandatory unique id.

   ```
   <tokenizer-plug-in id="my_new_tokenizer" library="my_plugin"
    set-up="pooling">
    <config>...</config>
   </tokenizer-plug-in>
   ```

   The `ID` attribute has to be a unique identifier, `library` must be set to the name of the plug-in library. The `set-up` attribute has to be set to the value indicated in the plug-in documentation. The `config` sub-element contains the configuration of the plug-in as described in its documentation.
3. Create a `tokenizer` element that makes use of the plug-in.

   ```
   <tokenizer language="xx">
    <plug-in name="my_new_tokenizer" />
    <normalization ... />
   </tokenizer>
   ```

   The mandatory `language` attribute specifies the codes of the language (the same ones as specified by the langid) for which the tokenizer shall be used. The sub-element `plug-in` must have the attribute `name` be set to the `ID` you have given to `tokenizer-plug-in` . If you are not sure which normalization parameters to use for a certain mode, it is safest just to copy those from the default tokenizer.
4. Follow the *Customizing the Tokenization Process* on page 27 to deploy the new tokenization configuration.

Exceptions

Exceptions are certain *exceptional* tokens that are usually separated, but can be kept as one token to improve retrieval precision.

Exceptions are gathered in an exception list and each exception consists of character sequences that have to be treated as one token. These exceptions override the configuration of the default tokenizer to some extent.

You can merge multiple tokens and add non-word characters to a token. However, exceptions will not be applied if it would split an already accepted token. For example, if an exception for the word *C++* was configured. The word *ABC++* would not be split, because it would split an already accepted token *ABC* .

Exceptions are matched against the surface form of the text. Meaning, the text after the application of the input-normalization. Therefore, exceptions have to be specified in every casing they are supposed to match.

Exception lists are dictionaries of type entity (cf. *Editable Dictionary Types* on page 142) containing all the exceptional tokens. Do not tokenize the dictionary.

Example

The word *C++* is usually split into the three tokens *C* , + and +. The word *.Net* is usually split into two tokens, . and *Net*. Since the tokens have more meaning kept together, an exception can be created to make sure those two strings are kept together.

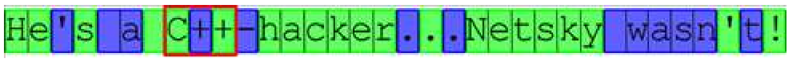


Figure **1**: Exception is located

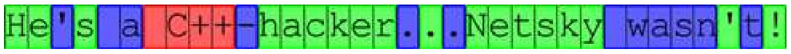


Figure **2**: Accept the exception since it merges tokens completely

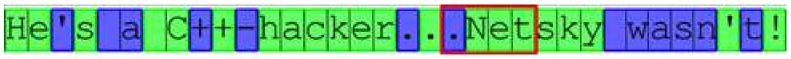


Figure **3**: *.Net* rejected in this context as exception since it would split the token **Netsky**

Add an Exception

This topic describes how a dictionary with exceptions is added to the tokenization process.

1. Make a copy of your currently used configuration, `$FASTSEARCH/etc/tokenizer/tokenization.xml`.
2. Add a `tokenization-exception` element within the `tokenizers` element and include the mandatory `automaton` element that specifies the untokenized entity dictionary containing your exceptions.

```
<tokenizers>
 <tokenization-exceptions>
  <automaton filename="path/to/dictionary/exceptions.aut" />
 </tokenization-exceptions>
</tokenizers>
```

Note: The path ot the automaton is relative to `$FASTSEARCH` and you have to deploy the file manually to all nodes.

3. Follow the *Customizing the Tokenization Process* on page 27 to deploy the new tokenization configuration.

Tokenization Modes

Tokenization Modes allow alternative, named set ups of the tokenization service. Modes enable different tokenization behaviours within one ESP installation.

This is very powerful and wrong configurations can result in severe problems. Consider contacting FAST Solution Services for advice.

Usage example

You have database field containing product names such as *my$Product* which contain non-word characters which are significant for the names, and where you want search case sensitive. You might want to set up a special tokenization for this field. This can be achieved by defining a tokenization mode which is applied only to this specific field, both when processing the field and when sending queries to this field.

The tokenization configuration has always one implicit mode: the default mode. The default mode has no ID and is made up by the `default-tokenizer` and `tokenizer` elements. As for any other mode the *default tokenizer* is mandatory for the default mode.

Add a Mode

This topic describes how to add a tokenization mode by following an example.

Assume the following default mode prior to adding the new mode:

```
<tokenizers>
 <generic-tokenizer id="fds4-compat" >
  <skip  name="compat-skip" />
  <split name="compat-split" />
 </generic-tokenizer>

 <!-- Declaration of the default mode -->
 <default-tokenizer>
  <generic name="fds4-compat" />
  <normalization transformation="lowercase" canonical="true" />
 </default-tokenizer>
</tokenizers>
```

1. Make a copy of your currently used configuration, `$FASTSEARCH/etc/tokenizer/tokenization.xml` .
2. Create a new `generic-tokenizer` "My_Tokenizer" under `tokenizers` .
3. Create the `mode` element in `tokenizers` and assign it the id "My_Mode". In the new mode element place the new default tokenizer that uses the desired generic tokenizer and normalizations.
   The new mode `My_Mode` should use a different generic tokenizer called "My_Tokenizer" that has to be declared first. Also the default tokenizer of the new mode should use the transformation lowercase-accentremoved to generate the canonical representation of the indexed tokens. The minimal result could look like the following:

   ```
   <tokenizers wordfolder_compatible="true" >
    <generic-tokenizer id="fds4-compat" >
     <skip  name="compat-skip" />
     <split name="compat-split" />
    </generic-tokenizer>

    <generic-tokenizer id="My_Tokenizer" >     <!--  New: tokenizer for new mode -->
     <skip  name="My_Skip" />
    </generic-tokenizer>

    <!-- Declaration of the default mode -->
    <default-tokenizer>
     <generic name="fds4-compat" />
     <normalization transformation="lowercase" canonical="true" />
    </default-tokenizer>

    <!-- Declaration of the mode "My_Mode" -->
    <mode id="My_Mode" >
     <default-tokenizer>
      <generic name="My_Tokenizer" />
      <normalization transformation="lowercase-accentremoved" canonical="true" />
     </default-tokenizer>
    </mode>
   </tokenizers>
   ```

4. Follow the *Customizing the Tokenization Process* on page 27 to deploy the new tokenization configuration.

Input Normalization

The input normalization is language independent and is applied prior to tokenization.

Input normalization is intended to bring the input text into a unified format without changing its meaning. Input normalization can include:

- Producing the canonical composed form of accented characters
- Building the canonical form of complex characters, such as ligatures
- Replacing certain less used Unicode character by other compatible characters or sequences of other characters (such as the Trademark sign with the string *tm*)

Note: All queries and filters are also normalized prior to further processing by the so called query normalization, which is identical to the input normalization.

The input normalization is typically a standard Unicode normalization (such as NFKC or NFC) that maps compatible characters and orders diacritics in a canonical way. It aims to not change the semantics of the content. The standard input normalization in FAST is NFKC.

Note: FAST generally does not support customization of the input normalization.

Character Normalization

Normalization is the process of reducing a character's complexity, either by changing or removing parts of it.

Documents that include accents or special characters can be routed through a normalization process during document processing.

If words are indexed with accents, like the french name *Côte d'Azur*, a query must include the accent if the document should be returned. If however the word is normalized, meaning the accent is removed, the document can be returned without any accents in the query string.

It is also possible to both normalize a character and keep the original. Multiple variants of one word can be stored using the word stacking feature for scope fields. Additional variants are added by adding more normalization transformations to the tokenizer process.

Normalization of Accents and Special Characters

Character normalization, such as accent removal for French, sharp-s conversion for German, and Katakana to Hiragana conversion for Japanese can be configured in FAST ESP. Character normalization can be customized in the tokenization configuration.

Note: The default tokenization configuration file includes no normalization for accented characters.

There are two extra tokenizer configuration files that configure the tokenizer for character normalization: `etc/tokenizer/tokenization-english.xml` defines a very aggressive normalization strategy, which is suitable for installations where only English documents are indexed. However, this configuration file could be used on installations where only French documents are indexed as well. It removes all diacritics and normalizes all other non-ASCII Latin character included in Unicode, up to code point 255 to the corresponding ASCII characters.

The configuration file `etc/tokenizer/tokenization-european.xml` is suitable for a mix of European languages, in collections where diacritics might be used inconsistently. It removes most accents, normalizes German sharp s, but leaves German and Scandinavian umlauts intact.

In addition to the character normalizer integrated in the tokenizer, there is a pre-configured separate document processor stage called `CharacterNormalizer` . On the query side, the instance is named `characternormalizer` . These stages can be used to perform replacement or normalization of characters on fields that are not passed to the tokenizer, or before or after certain stages in the pipeline where the tokenizer cannot be situated. The characters that these stages replace are configured in `$FASTSEARCH/etc/character_normalization.xml` . The character normalizer has to be placed manually in the desired query or document processing pipelines.

Enable a New Normalization Strategy

This topic describes how you can enable a different normalization strategy.

Note: Apply this change to all nodes in a multinode installation. If documents are already indexed, restart document processors and the Query and Result server, and refeed the documents.

Attention: If you use lemmatization, and entries in you lemmatization dictionary are affected by the normalization, you have to recompile lemmatization dictionaries with the new normalization settings in order for lemmatization to work properly.

1. Rename `etc/tokenization.xml` to `etc/tokenization-orig.xml`
2. Rename the tokenization file containing the new strategy, for example `etc/tokenization-english.xml` to `etc/tokenization.xml`
3. Restart document processors and the Query and Result server

   If you want to enable highlighting, uncomment `juniper.matcher.wordfolder.type advanced` in `$FASTSEARCH/etc/config_data/RTSearch/webcluster/fsearch.addon` and restart RTS Search.

Variants

Variants are different forms of the same word, stored at the same index position using the word stacking feature (available for scope fields only). Multiple variants of one word can be indexed to increase precision or recall.

When using Word Stacking it is possible to select the desired level of normalization on a per query basis. This in turn enables the following advanced linguistics features:

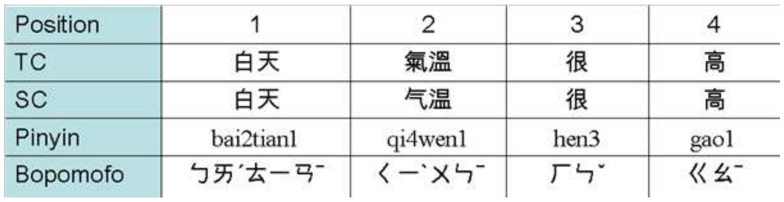
- Phrase or proximity (NEAR/ONEAR) queries in association with wildcards, lemmatization and character normalization (accent normalization)
- Per query selection of accent/case sensitive/insensitive matching
- Efficient handling of lemmatization in a multi-lingual environment. If you do not know the language of the query, and the index contains content in multiple languages, then a simple normalization to the base form (based on the default language setting) is not appropriate, as this can introduce ambiguities. Instead the linguistic query processing expands the query term to an OR between the original word and the base form of the word.
- Efficient handling of character normalization in a multi-lingual environment.

French publications will include all appropriate accents for French language words whereas English language documents containing French words often omit the accents. Suppose a French document contains the phrase **Côte d'Azur** and another English document contains the phrase **Cote d'Azur** without the accent. In the first document, **Côte** would be indexed as the two variants **côte** and **cote** . A user query of **Côte d'Azur** would hit only the first document (if selecting accent sensitive search), but a user query of **Cote d'Azur** would hit both documents.

A word is processed by a tokenizer and the tokenizer can specify one or multiple normalization transformations. Each transformation is responsible for transforming a word into a specific variant. Variants are identified by an unique name which allows explicit selection of a variant in a query.

```
<transformation name="lowercase">
 <fast name="lowercase" />
</transformation>
```

Note: Variants are only available for scope fields in ESP 5.1 and variant selection is only possible using FQL.



Position	1	2	3	4
TC	白天	氣溫	很	高
SC	白天	气温	很	高
Pinyin	bai2tian1	qi4wen1	hen3	gao1
Bopomofo	ㄅㄞˊㄊㄧㄢˉ	ㄑㄧˋㄨㄣˉ	ㄏㄣˇ	ㄍㄠˉ

FAST ESP is shipped with three predefined transformations, but additional transformations can be added.

- lowercase
- lowercase and remove all accents except diaereses and the ring above
- NFKC

Add a Variant

This topic describes how to make the tokenizer process add an additional variant of a word.

Adding a variant implies nothing more than adding a normalizer transformation to the tokenizer configuration. To make a normalizer available, edit the tokenization configuration and make sure the normalizer is declared as a `/configuration/transformations/transformation` element. Each transformation element needs a unique id in the `transformation/@name` attribute.

1. Make a copy of your currently used configuration, `$FASTSEARCH/etc/tokenizer/tokenization.xml` .
2. Make sure the normalizer transformation is declared as a transformation element and that it has a `name` attribute with a unique value.

   ```
   <transformation name="My_Normalizer">
    <fast name="lowercase" />
    <normalization><in>[Fast]</in><out>[FAST]</out></normalization>
   </transformation>
   ```

3. Add the new transformation as a normalization to the according tokenizers.
   For our example we only change a default-tokenizer. Generally you might want to use the same normalization setting for all tokenizers of one mode consistently.

   ```
   <default-tokenizer>
    <generic name="factiva-generic" />
    <normalization transformation="lowercase" canonical="true" />
    <normalization transformation="My_Normalizer" />
   </default-tokenizer>
   ```

4. Follow the *Customizing the Tokenization Process* on page 27 to deploy the new tokenization configuration.
5. After processing documents with the new settings, you can select the variants in scope fields by using the string parameter 'variant', `xml:string("FAST", variant="My_Normalizer")`

Character Normalization Independently from Tokenization

In addition to character normalization in the tokenizer, there are the separate document and query processors `CharacterNormalizer` and `qt_characternormalizer` that can be used to normalize document fields or query terms independently from the tokenizer if this is needed.

Note: Those stages are mainly included for backwards compatibility. For most purposes it is more advisable to use character normalization in the tokenization configuration instead, as this procedure is less error prone.

The separate character normalizers can be used to normalize fields for which no tokenization is wanted or for which normalization should be carried out prior to tokenization or later in the pipeline. It offers the functionality to replace any unicode character sequence by any other unicode character sequence.

The query transformer can also be used to add variants of terms to a query, e.g. for transliteration purposes.

- For both the document processor and the query transformer, the fields to be normalized can be specified. Source and target fields can differ.
- If used for character normalization, the document processor and the query processor should be used for the same fields to avoid recall problems.
- The query processor can also be used independently to add variants for a term. This can be used for transliteration search.

The characters to be normalized can be specified in the configuration file `etc/character_normalization.xml` . The default configuration will deaccent accented characters in western scripts and also replace some other characters, such as German *sharp s* or Russian *yo* , by normalized characters or character sequences.

Configuration File Format for the Character Normalizer

In the tag `normalization_data` this xml file contains `normalizationlist` elements. `normalizationlist` elements do not have any effect on the normalization process, they are only a means to organize the configuration file for better maintainability. Each `normalizationlist` element contains a set of `normalization` elements, which define the actual normalizations. Each normalization can have a `description` attribute, which only serves for better maintainability. The actual content of the normalization is defined in the `input` and and `output` element. Both input and output can be:

- A sequence of hexadecimal unicode references, separated by blank (e.g. `xE0`)
- A sequence of decimal unicode references, separated by blank (e.g. `196`)
- A string of UTF-8 characters, enclosed by square brackets, (e.g. `[c++]`)

The output can be empty - i.e. you can use the character normalizer to remove letters or letter sequences. One application of this feature is the removal of Arabic or Hebrew vowel marks.

Note: If a substring submitted to the character normalizer matches multiple input declarations, only the output for the longest match is used.

Defining two mappings of accented characters to unaccented ones.

```
<?xml version="1.0"?>
<normalization_data>
  <normalizationlist name="latin_accents">

    <normalization description="a breve to a">
      <input>xE0</input>
      <output>x61</output>
    </normalization>
    <normalization description="a circonflex to a">
      <input>xE2</input>
      <output>x61</output>
    </normalization>

  </normalizationlist>
</normalizationdata>
```

Mapping to a sequence of output characters

```
<normalizationlist name="german_sharp_s">
   <normalization description="sharp s to double s">
     <input>xDF</input>
     <output>x73 x73</output>
   </normalization>
</normalizationlist>
```

Mapping from a UTF-8 string to another UTF-8 string

```
<normalizationlist name="cmappings">
  <normalization description="c++ to cplusplus">
    <input>[c++]</input>
    <output>[cplusplus]</output>
  </normalization>
</normalizationlist>
```

Mapping defined in decimal unicode references

```
<normalizationlist name="Old to new Kanji">
    <normalization description="Some Kanji">
        <input>20315</input>
        <output>20175</output>
    </normalization>
</normalizationlist>
```

Mapping an Arabic vowel mark to nothing (removal)

```
<normalizationlist name="Arabic vowel marks">
   <normalization description="Fatha to nothing">
     <input>x64e</input>
     <output></output>
   </normalization>
</normalizationlist>
```

Multiple mappings for one character

This example covers from Latin script to Cyrillic (Russian). Such a configuration is only available for the query processor, and can for example be used for transliteration of queries, where multiple target transliterations are possible. Multiple mappings should never be used for the document processor!

```
<normalizationlist name="Latin to Cyrillic transliteration">
       <normalization description="a to cyrillic equivalents">
          <input>[a]</input>
          <output>[ ]</output>
          <output>[ ]</output>
      </normalization>
</normalizationlist>
```

CharacterNormalizer Document Processor

The document processor for tokenization independent character normalization is called `CharacterNormalizer`

Parameter	Description	Default value
attributes	Attributes that should be investigated and content should be normalized if necessary. Use the syntax field:field1 to put the normalized content into another field than the source field.	elemtitle elemheadings elembody xml
configfile	The characters to be normalized are specified in this configuration file.	`$FASTSEARCH/etc/character_normalization.xml`

Character Normalizer Query Transformer

`qt_characternormalizer` is the query processor for character normalization.

In the standard setup, any instance of `qt_characternormalizer` works on string nodes. Any instance must be placed after `parsefql` and before `tokenize` in the query processing pipeline to take effect. It can be configured to work on tokens (see below). In this case it has to be placed after `tokenize`.

The behaviour of `qt_characternormalizer` can be controlled by a number of parameters settings, both query parameters defined for each query and static parameters defined for a specific instance of the QT.

Query parameters

Those parameters control the behaviour of qt_characternormalizer on a per query basis. If you have a new stage based on qt_characternormalizer, use qt_YOURSTAGENAME instead of qtf_characternormalizer: as prefix.

Query parameter	Valid values	Description	Default
qtf_characternormalizer:normalize=value	true, false	Switches on or off query side character normalization.	depends on static parameter on_by_default

Static parameters

Those parameters are defined in qtf-config.xml for any stage based on qt_characternormalizer

Static parameter	Valid values	Description	Default
on_by_default	0,1	If set to 0, this stage is not active by default (note that default value is 1), but must be explicitely switched on using the `qtf_characternormalizer:normalize` parameter.	**1**
config_file	CONFIG_FILE_PATH	Configuration file path should be a value relative to $FASTSEARCH.	no default value
process_tokens	0, 1	Process token nodes instead of string nodes. Necessary if the stage is should work after the tokenizer (e.g. for post-normalization of output from synonym expansion).	0

Chapter 3

Lemmatization

Topics:

The purpose of lemmatization is to enable a query with one word form to match documents that contain a different form of the word.

In English, lemmatization can occur for:

- singular or plural forms for nouns,
- positive, comparative, or superlative forms for adjectives,
- tense and person for verbs,

For other languages, lemmatization also allows search across case and gender forms and other form paradigms, depending on the grammatical features for the word forms.

Lemmatization allows a user to search for a term like *car* and get both documents that contain the word *car* and documents that contain the word *cars*.

Lemmatization, stemming and wildcard search

Lemmatization differs from stemming or wildcard search by being more precise. Different word forms are mapped to each other by using a language specific dictionary, not by applying simple suffix chopping rules (stemming) or partial string matches (wildcard search).

In contrast to stemming or wildcard search, which would match all documents containing words starting with *car*, such as *cared* or *career*, lemmatization allows for recognizing words as matching terms on the basis of their being inflectional variations of the query word. With this, lemmatization also takes irregular inflections such as *tooth* and *teeth* into account.

Supported Languages for Lemmatization

Lemmatization is available for the following languages:

Note: Lemmatization dictionaries not shipped with the product can be requested. Contact your FAST Account Manager or FAST Technical Support for details.

Shipped with product	On request	On request
• Dutch	• Arabic	• Polish
• English	• Czech	• Romanian
• French	• Danish	• Russian
• German	• Estonian	• Slovak
• Italian	• Finnish	• Swedish
• Japanese*	• Hebrew*	• Turkish
• Korean*	• Hungarian	• Ukrainian
• Norwegian	• Hindi	
• Portuguese	• Latvian	
• Spanish	• Lithuanian	

Note: For languages marked with star (*), lemmatization is handled in the tokenizer. The lemmatizer only writes lemmas to the correct document fields and handles query term rerouting.

Lemmatization Strategies

You can choose from 3 modes of lemmatization:

- Lemmatization by document expansion
- Lemmatization by query expansion
- Lemmatization by reduction

Lemmatization by document expansion is the default strategy.

All lemmatization strategies have basically the same effect on the search experience, i.e. the set of documents returned for a query is the same, whatever strategy is used. Only in case of cross-language effects there might be slight difference in the returned document sets between lemmatization by document expansion and the two other strategies.

Lemmatization By Document Expansion

The words in documents are expanded to all existing full forms. There is no lemmatization of the query, but query terms are rerouted to match terms in the lemma index.

Lemmatization by document expansion is the default type of lemmatization for Western languages for FAST ESP. It is not recommended to use Lemmatization by expansion for any language that has a very high number of word forms per word, such as Finnish, Hungarian or Turkish.

Lemmatization By Query Expansion

The words in the query are expanded to all existing full forms. There is no lemmatization of the documents.

If no language is set on the query side, the default language is assumed. Nothing is done on the document side.

Lemmatization By Reduction

The words in the documents and in the query are reduced to their lemmas (canonical forms).

Only the base forms and the original forms are added to the index. If no base forms exist, then the original form is added to the lemmatized index. Lemmatized query terms are sent to the index of lemmatized words.

Note: Lemmatization by reduction is the only available mode for Japanese and Korean.

Note: Lemmatization by reduction can be recommended for languages with a very high number of word forms per word, such as Finnish, Hungarian and Turkish.

Comparison of Lemmatization Strategies

The following table gives an overview of the features for the different lemmatization strategies.

	Lemmatization By Document Expansion	**Lemmatization By Query Expansion**	**Lemmatization By Reduction**
Disk space consumption	High, especially for languages with high count of forms per word, such as Russian or Finnish.	No additional disk space consumption (index)	Low due to smaller dictionaries.
Impact on document processing or query processing	Apart from the rerouting of the query term, only document processing is affected.	No impact on document processing. Query expanded with additional word forms. is affected.	Document processing and query processing must be synchronized.
Language	Independent of the language of the query term.	Query language is pre-selected or known.	Query language is pre-selected or known.
Implicit and explicit proximity	Supported for scope fields. For normal fields, lemmatization by document expansion does not support implicit and explicit proximity.	Supported for scope fields and normal fields.	Supported for scope fields. For normal fields, lemmatization by reduction partially supports implicit and explicit proximity.

If you cannot afford much disc space and/or use only a single language, use lemmatization by reduction. Otherwise, lemmatization by document expansion is generally recommended, especially in cases where more than one language is involved.

Note: It is not possible to change the mode of lemmatization on a per query basis. Make sure that lemmatization is configured with the same mode on document processing and query processing side. Otherwise queries will not be handled correctly.

Refer to *Change Lemmatization Strategy for a Language* on page 50 for details how to change the lemmatization modes.

Language Identification for Lemmatization

The choice of lemmatization dictionaries for document processing relies on the language of the document. Therefore lemmatization can only be used in conjunction with the language identifier or if the language of the document is set by some other pipeline stage.

All standard document processing pipelines that include lemmatization also include language identification.

For lemmatization by reduction, the query language has to be known. In most cases, it is not possible to identify query languages automatically because queries are too short. Therefore the query language has to be unambiguous or set manually for lemmatization by reduction.

Lemmatization Dictionaries

Lemmatization uses two types of dictionaries:

- Dictionaries containing the forms and base form for each word, referenced as `Lemmatization dictionaries`.
- Blacklist dictionaries that allow to exclude certain lemmatizations without editing the main dictionary, referenced as `lemmatization stopword dictionaries`.

Lemmatization Dictionaries

Lemmatization dictionaries can contain word forms for nouns, adjectives and verbs. The choice of the lemmatization dictionary for a language provides a way to select which parts of speech should be subject to lemmatization. That means you are able to select the level of lemmatization. The default level is normalization of nouns and adjectives (NA). The following dictionaries, in `$FASTSEARCH/resources/dictionaries/lemmatization` are available per language.

Dictionary	**Supported lemmatization mode**	**Filename**
Nouns only (N)	Reduction and expansion	`_N_red, _N_exp`
Nouns and adjectives	Reduction and expansion	_NA_red , _NA_exp
Nouns and verbs	(on request)	_NV_red
Nouns, adjectives and verbs	Reduction and expansion	_NAV_red, _NAV_exp

Note: Verbs are not available for all languages. Contact Solution Services for a list of which languages include verbs.

Tip: Contact FAST Customer Support if you need other variants of the dictionaries. The only dictionaries that cannot be inspected or configured for different parts of speech are dictionaries for Chinese, Japanese and Korean.

Choosing Lemmatization Dictionaries

Information on which word categories should be lemmatized is determined by what dictionary is chosen. This topic describes which types of lemmatization dictionaries are appropriate for which task.

The choice of dictionary influences indexing time, index size and query processing time (depending on the lemmatization strategy). Using a dictionary with all possible forms will increase the number of index terms.

Using a dictionary that covers more parts of speech (nouns, adjectives an verbs) can lead to ambiguities and will lower the precision. For example, if your query contains the noun *download* and a lemmatization dictionary for nouns and verbs is used, it will not only be expanded to *downloads* , but to *downloading* and *downloaded* as well, since *download* could also be the verb in present tense.

As a general rule, it is recommended to use the noun or noun and adjective dictionaries for generic applications. Dictionaries with verbs are more useful for applications whose content is action-centered, where verbs carry a lot of semantic information. An example for this may be a user help-desk, where users often will use verb phrases to describe their problems. In this case, it can be useful to expand the verbs into their alternative forms as well.

The following paragraph gives a short example of the effects of the different dictionaries. Assume that the original document contains the phrase:

This download allows developers to run quick tests on our server.

Then the following word forms will be matched in addition to the original content if the lemmatizer is used with the listed parts of speech combination:

Dictionary	Added word forms
N	downloads developer runs test servers
NA	downloads developer runs quicker quickest test servers
NV	downloaded downloading downloads allowed allowing allow developer running runs ran tested testing test servers
NAV	downloaded downloading downloads allowed allowing allow developer running runs ran quicker quickest tested testing test servers

Lemmatization Blacklist Dictionaries

The `lemmatization stopword dictionaries` include words that should not be considered for lemmatization.

In addition to the core lemmatization dictionaries, it is possible to define lemmatization stopword dictionaries. They contain words or phrases that are to be excluded from lemmatization.

If you set up a stopword dictionary which contains the phrase *Fast Search and Transfer*, the `Lemmatizer` document processor will skip this phrase during document processing, and this text will not be retrieved when searching for *searches* or *transfers*

Note: Multiterm lemmatization stopwords or only considered during document processing, and therefore cannot be used with lemmatization by reduction and lemmatization by query expansion. You can switch off lemmatization for individual query strings for all modes by using the string parameter `linguistics=off`

Note: Lemmatization stopword dictionaries are not query stopword dictionaries! Words that are listed in the lemmatization stopword dictionary are excluded from lemmatization, but not from the query string as such. That means that these words are not lemmatized, but still mapped to the search index.

Configuring Lemmatization

Lemmatization can be configured to operate during document indexing and/or during query processing.

Note: `LinguisticsStudio` can be used to configure lemmatization for both document indexing and query processing.

Note: The configuration options listed in this chapter do not apply to Chinese, Japanese, Korean or Thai.

Lemmatization in the Index Profile

The index profile is where you can indicate for which fields lemmatization is carried out in FAST ESP. Using `LinguisticsStudio`, lemmatization can be configured for an installation without having to modify the index profile manually.

Two attributes are available to steer lemmatization:

- `lemmatization=yes|no`: (normal fields and scope fields only): Run the lemmatizer for this field.

- `lemmas=yes|no`: (composite fields only): Configures whether lemmas from the fields that constitute this composite field are put in the index.

If you are using lemmatization by query expansion, you can set up lemmatization independently of the index profile and configure the query transformer for any field in the index. For this purpose you have to edit the field specification for the lemmatizer in `qtf_config.xml` manually. However it is recommended to use the index profile method for this case as well.

Lemmatization Configuration File

The central configuration file for lemmatization for both the query side lemmatization and document lemmatization is `$FASTSEARCH/LemmatizationConfig.xml`. This file contains the following information:

- A definition of lemmatizers for all languages handled by the lemmatizer proper. They are defined as `standard_lemmatizer`. Each lemmatizer has the following properties
  - `language`: the language for which this lemmatizer is defined
  - `mode:`: the lemmatization mode, can be `document_expansion, query_expansion` or `reduction`
  - `active`: allows to switch on/off the lemmatizer for a languages without removing it from the configuration file.
  - `lemmas` (subtag): defines the parts of speech (`N`,`NA` or `NAV`)

  The correct dictionary is automatically chosen based on this information.
- The definition of the default query language, i.e. the language that is assumed for a query where the language parameter is not set. This information is set in the `lemmatization` tag.
- The optimization: `qps` for maximum query throughput (default) and `indexsize` for smaller indexes (disables phrase support for lemmatization by reduction)
- A declaration of the languages lemmatized by the tokenizer. For those languages, the lemmatizer does not produce base forms, but writes the base forms supplied by the tokenizer to the appropriate document fields, and reroutes query lemmas. They are declared as an `external_lemmatizer`. Here only the language code has to be set.

Note: Any change to the lemmatization configuration requires reprocessing of all documents, except for lemmatization by query expansion.

Here an extract from `LemmatizationConfig.xml`:

```
     <lemmatization default_mode="document_expansion" optimization="qps"
 default_query_language="pt">

      <standard_lemmatizer language="es" mode="reduction" active="yes">
         <lemmas active="yes" parts_of_speech="NA" />
      </standard_lemmatizer>

      <standard_lemmatizer language="fr" mode="document_expansion"
active="yes">
         <lemmas active="yes" parts_of_speech="NA" />
      </standard_lemmatizer>

      <!-- Third-party solutions with lemmatization in the tokenizer -->
     <external_lemmatizer language="ja" description="Japanese lemmatizer"
 />
      </lemmatization>
```

Lemmatization During Document Processing

The document processor for lemmatization is called `Lemmatizer`. A autoconfigured instance of the processor, e.g. `Lemmatizer (webcluster)` is created for each cluster based on the index profile. All lemmatization stages are situated after the tokenization stage (tokenizer). This order must not be changed, as the lemmatizer requires pre-tokenized input.

Lemmatizer
The `Lemmatizer document processor` is used to lemmatize specified fields according to a configuration file.

Lemmatization is configured in the `configfile`. Lemmatization is performed for documents with the languages that are defined in the configuration file, and for the document attributes that are defined in the processor's parameter `attributes`. The lemmatizer stage must be placed after the `tokenizer` in order to work properly, because it is configured to process tokenized content.

Parameter	Description	Default value
attributes	A space separated list of fields/elements which content should be lemmatized. The lemmas will be stored in the same field, unless a mapping on the form `source:target` is supplied. In the latter case, the lemmas are written to the `target` field. If using lemmatization by document expansion, the original text is also added to the output field. It is possible to mix the two formats. This parameter is auto-configured in the `Lemmatizer` stage for each cluster, based on the index profile.	title headings body
configfile	The `configfile` parameter defines the name of the configuration file for the mappings of languages on dictionary resources; see the comments in that file for further details.	$FASTSEARCH/etc/LemmatizationConfig.xml

Lemmatization During Query Processing

The query processor for lemmatization is called `lemmatizer`. The The lemmatization stage is situated after the tokenization stage (tokenizer). This order must not be changed, as the lemmatizer requires pre-tokenized input. The `lemmatizer` is autoconfigured based on the index profile. For lemmatization by query expansion, the configuration can be changed, for other lemmatization strategies, query lemmatization has to operate on the same fields as document lemmatization. Per default, the query lemmatizer stage is configured to use the same configuration files as the document processor `Lemmatizer`, `LemmatizationConfig.xml`. If the configuration file name or path is changed, this should be done on the both query and document side.

Change Lemmatization Strategy for all Languages

You want to change the lemmatization strategy for all languages.

Resolution:

All steps apply to the configuration files `$FASTSEARCH/etc/LemmatizationConfig*.xml` and should be performed on all nodes. This procedure assumes that you have been using lemmatization by expansion, the default strategy, and you want to change to one of the other strategies. It also assumes that you have not changed the original lemmatization setup after installation of ESP. If you have changed the setup, you might need to edit the configuration files.

Note: This is a shortcut to change lemmatization strategies quickly for all languages. You can achieve the same goal be editing `$FASTSEARCH/etc/LemmatizationConfig.xml` directly.

1. Move the original file `$FASTSEARCH/etc/LemmatizationConfig.xml` to `$FASTSEARCH/etc/LemmatizationConfigDocumentExpansion.xml` for backup reasons.
2. Rename the file `$FASTSEARCH/etc/LemmatizationConfigReduction.xml` to `$FASTSEARCH/etc/LemmatizationConfig.xml` if you want to use lemmatization by reduction, or rename `$FASTSEARCH/etc/LemmatizationConfigQueryExpansion.xml` if you want to change to lemmatization by query expansion.
3. Restart both document and query processor.

Change Lemmatization Strategy for a Language

You want to change the lemmatization strategy for a language.

Resolution:

All steps apply to the configuration file `LemmatizationConfig.xml` and should be performed on all nodes.

1. You will need the lemmatization dictionaries for the language you want to install.
   Before requesting additional packages, check to see if the required dictionaries are already present in `resources/dictionaries/lemmatization`. If you need dictionaries for additional languages, contact FAST Solution Services.
2. Search for the `standard_lemmatizer` item for the language you want to change by following the example in the configuration file. Set the mode to `query_expansion`, `document_expansion` or `reduction`.
3. Change the parts_of_speech to the desired level.
4. Restart qrserver and procserver.
5. Setting up a new language for lemmatization by reduction or by document expansion requires re-processing of all documents. Re-process all documents.

Add a language for lemmatization

You want to add a language for lemmatization.

Note: Before requesting a language pack, you should check whether the language is already available. Refer to the table of supported languages (*Supported Languages for Lemmatization* on page 44) for a list of pre-installed and available language packs. For lemmatization, a special license is needed.

All steps apply to the configuration file `LemmatizationConfig.xml` and should be performed on all nodes.

1. You will need the lemmatization dictionaries for the language you want to install.
2. Add the language as a `standard_lemmatizer` item, following the example in the `LemmatizationConfig.xml` configuration file:
   ```
   <standard_lemmatizer language="es" mode="document_expansion" active="yes">
     <lemmas active="yes" parts_of_speech="NA" />
   </standard_lemmatizer>
   ```
   Set the mode to `query_expansion`, `document_expansion` or `reduction`, and configure the parts of speech.

   The `parts_of_speech` describes the sequence of Nouns (N), Adjectives (A), Verbs (V).
3. Restart qrserver and procserver.
4. Setting up a new language for lemmatization by reduction or by document expansion requires re-processing of all documents. Re-process all documents.

Set up default query lemmatization languages

You want to set up default query lemmatization for one or multiple languages.

By default, one query language is configured as the default query language for lemmatization. The lemmatizer for this language is used if no language tag is set for a query. You can change the default query language, and you can also define multiple default query languages. In the latter case, the lemmatizers for all configured languages are applied to the query.

Note: This only applies to lemmatization by query expansion or by reduction. For lemmatization by document expansion, there is no query lemmatization apart from rerouting of query terms to the lemmatized content, so the choice of the default query language does not affect search results.

1. If not already done, define a `standard_lemmatizer` for each language. (Refer to the previous Lemmatization issues/resolutions.) If you want to use multiple languages as the default query language, all languages that are used in such a composite lemmatizer must be configured in the same lemmatization mode.
2. Add the desired default query language or a comma separated list of languages in the `lemmatization` tag in the `default_query_language` attribute as `default_query_language` :

```
<lemmatization default_mode="reduction" default_query_language
      ="en,de">
```

3. Restart the qrserver.

Interaction with other Modules and Features

Lemmatization and Synonyms

The effect of synonym expansion on lemmatization depends on the lemmatization strategy

For more information, see the section on *Synonyms and Lemmatization* on page 69 in the chapter about synonymy.

Lemmatization and Wildcards

Wildcard search and lemmatization are mutually exclusive.

If wild cards are used in a query (e.g. searching for *comput** to match *computer* and *computing*), the query will not be lemmatized and will always be routed to the index without lemmas.

Lemmatization, Tokenization and Character Normalization

This section only applies if you are using lemmatization and character normalization in the system, or if you have changed other tokenization configuration settings. If you have changed the tokenization configuration on your system, or you have set up character normalization, e.g. mapping of accented character to unaccented ones, you need normalized lemmatization dictionaries for all languages where characters are affected by the changes. If you remove accents, for example, you will have to use a normalized French lemmatization dictionary. There are two ways to get the dictionaries:

- Recompile the dictionary sources using the dictionary compilation tools. The dictionary sources for the standard languages are included in the product.
- For languages where the dictionary sources are not installed on your system, contact your FAST Account Manager.

Lemmatization and Tokenization

The lemmatizer, both on the query and document side, receives the content that has been processed by the tokenizer. The input is therefore lowercased and normalized according to the tokenizer configuration. The compiled dictionaries shipped with the product are tokenized based on the standard tokenizer configuration. Lemmatization dictionaries edited with the dictionary editing tools are automatically tokenized using the tokenizer configuration.

Lemmatization and Character Normalization

If character normalization is configured in the tokenizer configuration (the recommended method), this is handled as described in the previous section.

If you set up character normalization using the separate character normalizer stage, you have to set the property `compWithCharNormalize` to `true` in the dictionary editing tools and recompile the lemmatization dictionaries.

Lemmatization With Proper Names and Phrases

In the standard configuration, lemmatization and proper name recognition cannot be applied on the same query term at the same time.

For normal fields if you are using the default lemmatization strategy, lemmatization by document expansion, lemmatization will not be applied to query terms that are marked as phrases, or which are joined by proximity operators. This also holds for complex proper names recognized by the `didyoumean` spell checking framework. These terms are matched only against the usual search index. For example, *FAST Search* may be included in the list of proper names, which would exclude the inflections *fasts* and *searches* in the lemmatized index.

If you use lemmatization by reduction, phrases will be lemmatized.

For scope fields, lemmatization is available for all type of queries in all lemmatization modes.

When proper name, phrase recognition, and lemmatization are applied simultaneously to a query, proper name and phrase recognition will override lemmatization under the specified conditions. Thus, if your Search Front End provides both lemmatization and proper name and phrase recognition not as mutually excluding functionalities, but as options that can be selected simultaneously, it is recommended to provide these two selections as mutually excluding radio buttons on your Search Front End.

Lemmatization and Proximity Operators

Normal (non-scope) fields do not support lemmatization with proximity operators - the lemmatization setting is ignored and you will only get correct matches for the original terms in the document. When using FQL you will also get Query Tranformation feedback indicating lemmatization settings were ignored.

Compound segmentation

In some languages, such as German, the Scandinavian languages, Dutch or Korean, compounds are written in one token without spaces. For example, the English term *search engine user* can be written as one word in German, *Suchmaschinenbenutzer*. To achieve better recall, such complex words can be split up in their components through the process of compound segmentation. This feature is not available in the standard ESP package (apart from Korean) but can be added.

- For Korean, this feature is configurable. Refer to the chapter *Processing Content in Asian Languages* on page 99 for more details
- For German, an extra package is available to enable this feature. Contact your FAST Account Manager for details

Chapter

4

Spellchecking Framework

Topics:

The spell checking algorithm operates on individual query segments. A query segment is a portion of the query that forms a syntactical entity of some kind. For example, if an item within the query is put in quotes, that quoted part forms a query segment.

Checking the spelling of a query is executed in two stages: First, an Advanced Spell Check is performed, including phrase recognition, followed by a Simple Spell Check. In addition, the `didyoumean` query processor also includes word splitting and advanced stop word removal. The processor operates on string nodes and displays suggestions or modifications done to the string in the form of query transformation feedback messages.

Supported Languages for Simple Spell Checking

Spell checking dictionaries are available for the following languages:

Note: Spell checking dictionaries not shipped with the product can be requested. Contact your FAST Account Manager or FAST Technical Support for details.

Shipped with product	On request	On request
• Dutch • English • French • German • Italian • Korean • Norwegian • Portuguese • Spanish	• Arabic • Czech • Danish • Estonian • Finnish • Hebrew • Hungarian • Hindi • Latvian • Lithuanian	• Polish • Romanian • Russian • Slovak • Swedish • Turkish • Ukrainian

Spellchecking Stages

By default, the spell checking framework, defined in the `didyoumean` query pipeline instance, performs the following stages for each submitted query, in order. Each stage can be turned on and off separately and can be configured to work in either rewrite or suggest mode.

1. Word splitting: A word which is not in the spellcheck dictionary, but can be split into two parts which generates a match in the spellcheck dictionary, is split up. For example, *systemreboot* is corrected to *system reboot*. By default, word splitting is based on the English spell checking dictionary. The dictionary to be used can be changed for each search profile in the Search Business Center.
2. Advanced phrase level spell checking and phrase recognition: Any proper names (and other phrases) in the query are identified, phrased and corrected using a proper name dictionary. This dictionary is language independent. Phrasing means that the name parts are not searched as independent words any more, but only found if they are adjacent in the searched document. For example, when *William Shakespeare* is phrased, the misspelled term *William Shakespaere* will be corrected and phrased to *"William Shakespeare"*
3. Spell checking on simple terms: Language specific spell checking is performed. Only single words are considered, not phrases. For example, the term *acctor* will be corrected to *actor* if the stage is actived and the query languages is set to English.
4. Anti-phrasing (also see next chapter): Phrases such as *what is* and *can you give me information on* do not contribute to search precision and are removed by the Anti-phrasing removal feature.

When all listed stages are completed, suggestions or modifications to all string nodes are aggregated and a modified query is generated and presented.

Advanced Phrase Level Spell Checking

During the Advanced Spell Check stage, the query terms are run through Advanced Phrase Recognition. In this stage, complex, multi-word terms are detected, phrased, and eventually corrected.

This stage combines phrase detection with spell checking:

- it offers corrections for misspelled multi-word terms.
- it provides the functionality of phrasing recognized multi-term expressions, where phrasing means that the terms only produce a hit if they occur as consecutive terms in the document.

Both functionalities can be configured as suggestions or query rewrites.

Note: For earlier FDS 4.x users: this was previously referred to as "Proper Name Recognition".

Advanced Spell Check Dictionaries

The following dictionaries support advanced spell checking:

- **phrase dictionaries**

FAST ESP supplies a phrase dictionary that contains common phrases such as names of famous persons (for instance "elvis presley"), names of places (for instance "san francisco"), and names of companies (for instance "kraft foods").

You may modify the supplied phrase dictionary by adding or removing terms. Alternatively, you may create separate phrase dictionaries that contain customer specific phrases only. If you choose to create multiple phrase dictionaries, you can enable selecting a specific phrase dictionary to be used for spell checking at query time.

If a query phrase does not exactly match any entry in the selected or default dictionary, but is close to some dictionary entries, the phrase that is considered the closest match is suggested as a replacement to the original query phrase. If a query phrase matches an entry in the dictionary exactly, the phrase is protected by quotes and Simple Spell Check will not be allowed to change the terms of the phrase. If there are no entries in the dictionary that are close to matching the query phrase, the query phrase remains unchanged.

- **the phrase exception list**

FAST ESP supplies a default phrase exception list that contains phrases that are not to be considered for spell checking. When a query term matches an entry in the exception list, the term will be protected from spell checking changes.

You can adapt this phrase exception list to suit your content.

Note: All phrase dictionaries are language-independent. Note however that the default phrase dictionaries supplied with FAST ESP are optimized for English.

Updating Advanced Spell Checking Dictionaries

The dictionaries used for Advanced Spell Checking can be either modified or replaced by other dictionaries.

Spell checking dictionaries edited with `LinguisticsStudio` are automatically deployed correctly. Also dictionaries modified with `dictman` are deployed to all search nodes and search profiles if you run `view-admin -m deploy -a`. The following only applies if you want to deploy dictionaries created with `dictcompile` manually.

Note: It is possible to use multiple dictionaries of type SPELLCHECK_PHRASES. For instance, you can use the FAST default dictionary together with one or more customer specific dictionaries. It is recommended that you contact FAST Solution Services to perform the necessary configuration changes to support multiple dictionaries.

1. For all nodes running the FAST Query and Result server, copy the binary version of the new/updated dictionary to `$FASTSEARCH/resources/dictionaries/spellcheck/ (Unix)` or `%FASTSEARCH%\resources\dictionaries\spellcheck\ (Windows)`
2. When adding a new dictionary, make the appropriate changes in `$FASTSEARCH/etc/config_data/QRserver/webcluster/etc/qrserver/didyoumean/propername.xml` and copy that file to all nodes running the FAST Query and Result server.
3. Restart all FAST Query and Result Server components to load the updated dictionary.

Spell Checking on Simple Terms

The Simple Spell Check stage supports spell checking of individual terms against language specific dictionaries. The dictionaries are selected based on the query language. If the query language is not set, the default language is used. This spell check stage will only detect misspelling of single words, not phrases. Simple spell checking does not protect the corrected terms from further processing.

Simple Spell Check Dictionaries

The following dictionaries support simple spell checking:

- **single word dictionaries:**

FAST supplies language specific dictionaries that contain common words for the particular language.

If a query term does not match any entry in the dictionary, but is close to some dictionary entries, the term that is considered the closest match is suggested as a replacement to the original query term. If a query term exactly matches an entry in the dictionary, or there are no entries in the dictionary that are considered close to matching the query term, the query term remains unchanged and is sent to the FAST Search Engine.

You may modify the supplied single word dictionaries by adding or removing terms.

- **single word exception lists:**

Single word exception lists are dictionaries that contain words that are not to be considered for spell checking. When a query term exactly matches an entry in the exception list, the term will be protected from Simple Spell Check.

Note: In contrast to the phrase exception list, the single word exception lists are language specific.

Updating Simple Spell Checking by Language

You can update the Simple Spell Checking dictionary for a given language. To activate the new dictionary:

1. For all nodes running the FAST Query and Result server, copy the binary version of the dictionary to `$FASTSEARCH/resources/dictionaries/spellcheck/ (Unix)` or `%FASTSEARCH%\resources\dictionaries\spellcheck\ (Windows)`
2. If the configuration file in `$FASTSEARCH/etc/config_data/QRserver/webcluster/etc/qrserver/didyoumean` has changed, copy that to all nodes running the FAST Query and Result server.
3. Restart all FAST Query and Result Server components to load the updated dictionary.

Spell Checking Configuration

Several configuration files allow to configure the spell checking framework in ESP.

The behaviour of advanced and simple spell checking is defined in the following configuration files, which are all located in `$FASTSEARCH/etc/config_data/QRServer/webcluster/etc/qrserver/didyoumean`. Some parameters can also be set in qtf-config.xml directly.

Not all parameters are listed here, only the parameters that are often changed to tune spell checking

- spellcheck.<lang>.xml, where 'lang' is the a language abbreviation configures simple spell checking for a particular language.
- propername.xml configures advanced spell checking.

Note: Changes to the configuration files require a restart of the qrserver.

Parameter	Values	Default	Description
tolerance	integer	2	The similarity of two terms to each other can be measured in terms of the number of edit operations (deletion, substitution, transposition, addition) that are necessary to get from one term to the other. For example, one edit operation (deletion) is necessary to get from *winndow* to *windows*. The parameter *tolerance* determines the number of edit operations that are allowed to get from one term to the other.
threshold	integer	195 (220 for phrases)	If a term in the query matches a term T in a spell check dictionary with a matching score equal to or above the threshold, T is a candidate for a correction. By lowering the threshold the number of terms that match a given term T can be increased. However, it is recommended not to use numbers below 195.
exact	yes/no	yes	If this parameter is set to `yes` a term T1 that is part of a spell check dictionary cannot be corrected by another term T2 in a spell check dictionary. This configuration should be chosen if the spell check dictionary is "clean", i.e. if one can be relatively sure that it does not contain any spelling errors. By setting the value to "no" the frequency of a term is given more weight in determining which correction to choose. This parameter setting should be chosen for web applications.

In addition, there are several parameters that can be set in the qtf-config.xml directly for each spell checking substage. The table below lists the setting for the English spell checking, the parameters for other languages and phrase spell checking are named accordingly.

Parameter	Values	Default	Description
simple.en.length.threshold	Integer	4	Determines how long a word (in terms of number of characters) has to be in order to be spell checked. The idea is that very short words should not get spell checked as this might easily lead to wrong corrections. But if you need to have words with less than 4 characters spell checked change this parameter accordingly.
simple.de.encoding	String	-	The encoding of the dictionary files. Spell checking requires single byte encodings.

Query Parameters for the Spell Checking Framework

Spell checking can be controlled at query time by a set of query parameters.

The behaviour of the `didyoumean` stages can be controlled either in unison or as separate instances at query time. For example, it is possible to disable all features but Anti-phrasing.

Each stage can operate in one of three modes:

Mode	Description
on	The query is modified automatically.
suggest	Query modification suggestions are returned to the user.
off	Feature is disabled

The following query parameters switch on/off the `didyoumean` and the different stages.

Note: Either use parameter *spell*, which controls all stages, or the parameters that control the individual stages, but do not combine both parameter types.

Query parameter	Valid values	Description	Default
spell=value	on, off, suggest	Controls all stages.	suggest
qtf_didyoumean:rewrite_pre=value	on, off, suggest	Controls word splitting	off
qtf_didyoumean:phrasing=value	on, off, suggest	Controls phrase level spell checking and phrasing	off
qtf_didyoumean:spellcheck=value	on, off, suggest	Controls simple spell checking	off
qtf_didyoumean:rewrite_post=value	on, off, suggest	Controls antiphrasing/stopword removal	off

The following query parameters control additional features of the spell checking framework.

qtf_didyoumean:addconsidered=value	on, off	Didyoumean only displays one suggestion. With this parameter, considered terms (i.e. other terms than the winning one which have been found as corrections) are returned.	off
qtf_didyoumean:consideredverbose=value	on, off	Considered terms are output with detailed score information. To be used together with the `addconsidered` parameter.	off

Note: You can also control any custom stages based on `didyoumean`, and also `didyoumean` instances in search profiles by using the named query parameters. You will have to replace `qtf_didyoumean` by `qtf_YOURSTAGENAME`, e.g. `qtf_didyoumean-myprofilesppublished` to control didyoumean in the published search profile `myprofile`.

Aligning Spell Checking Dictionaries and Content

Spell checking dictionaries should only contain words that actually occur in the content. Otherwise spell check suggestions might lead to zero hit result sets when clicked on.

The `didyoumean` feature is very much dependent on suitable spell check dictionaries in order to work properly. While the standard dictionaries provide good results for general www-oriented setups, there are installations where this is not sufficient. There might be highly site-specific terms (such as product names) that should be checked for correct spelling. On the other hand, certain terms that are present in the spell check dictionaries might not occur in the actual content. In this case, there should be no spell check suggestion for such a term.

Therefore ESP comes with a framework for adapting the spell check dictionary to the actual content that is present in the system. It creates language-specific spell check dictionaries based on the actual content of the indexer nodes. Thus, the `didyoumean` feature can always stay in sync with the terms that are present in the system even when document deletes or collection deletes occur which substantially change the content.

Aligning Spell Checking Dictionaries

The alignment process consists of two phases that can be executed independently of each other. Thus, the alignment preparation can be done during feeding pauses while the actual alignment can be carried out during times with low query rates in order to ensure optimal resource usage.

The description creates frequency data for English and German. Of course, other language combinations can be used.

1. Run `createspelldata -l de -l en -a` on all indexer nodes.

   Make sure that a significant portion has actually been indexed before running this command. Otherwise, the spell check dictionary alignment will not yield optimal results.

   Frequency data for German and English will be collected in a central database.
2. Do the actual alignment inside the `dictman` tool.

   Make sure that all `createspelldata` processes have finished before continuing.

   a) Start `dictman`.

   b) Execute `tunespellcheckfl SPW en_spell_iso8859_1 spelltuning_en 4` to align the English dictionary.

   c) Execute `tunespellcheckfl SPW de_spell_iso8859_1 spelltuning_de 4` to align the German dictionary.

   The spell check dictionary sources are now aligned.

   - They contain only words present in the content.
   - The weight of the terms is aligned to their frequency in the content.
   - Terms that do not occur in the index are removed from the dictionaries.

3. Now all views for search profiles that use these spell check dictionaries need to be redeployed. You can do that either through the `Search Business Center` or using the `view-admin` command.
   To redeploy all views, execute `view-admin -m deploy -a`

Parameter Overview for the Spell Check Optimization Tools

Spell check dictionary optimization can be tweaked with various settings.

The `createspelldata` Command

Option name	Short name	Description
--cachemanager	-a	The frequency information extracted from the index is stored in a central database. Thus it can be used from within the `dictman` tool. If this option is not given, the frequency lists are written to local files.
--prune	-r	The frequency information is kept in memory during the collection process. Under some circumstances, the available memory is not sufficient for this task. If this option is set, the frequency information is regularly pruned while the collection process is running to make sure it fits into the system memory. As a result of this, terms with very low frequency will be removed from the frequency lists. For most applications, this has no functional impact. If all terms are to be kept, the option --withshelve can be set to use a disk based store.
--withshelve		The frequency information is collected to a disk-based structure instead of in memory. This ensures that the collection process does not run out of memory but will significantly decrease the speed of the operation.
--collection	-c	Only take into account terms that occur in documents that are part of the specified collection. Several collections can be specified by using this switch multiple times.

Option name	Short name	Description
--word		This option specifies from which index field the terms are to be extracted. By default, only the `body` field is used. The fields must be specified with their index name, e.g. `bconbody` instead of `body`. If just a field name is given, this name is assumed to specify an index context. Summary fields can also be used for collection spell data by additionally specifying the field type (--word=bsumcompanies:sField).
--phrase		When running with output to local files (i.e. --cachemanager is not specified), `createspelldata` can also collect phrase information from fields with a separator (e.g. navigator fields). in this case, the field name, type and the separator have to be specified. For instance, to collect companies into a phrase dictionary: --phrase=sField:bsumcompanies:;.
--minlength		The minimum length of words that should collected. Usually, words that are shorter than three characters should not be used for spell checking and thus do not need to be collect. Restricting the minimum length can potentially save a lot of memory and processing time during the collection process. The default is to collect all words.
--maxlength		The maximum length of words that should collected. Restricting the maximum length can potentially save a lot of memory and processing time during the collection process. The default is to collect all words.
--withfilter		Apply several filter heuristics during the collection process to remove terms that do not look like natural language words (i.e. "aaa134").

The `tunespellcheckfl` Command in `dictman`

The `tunespellcheckfl` command in `dictman` aligns a spell check dictionary with a frequency list (which has been collected using `createspelldata`). The general format of this command is: `tunespellcheckfl type name freqlist threshold`

Parameter Name	Description
type	The type of the dictionary that should be aligend, usally `SPW` for spell check dictionaries.
name	The name of the dictionary that should be aligend, e.g. `en_spell_iso8859_1`.
freqlist	The name of the frequency list that should be used for aligning the dictionary. The frequency lists created by the `createspelldata` tool have the following name pattern`spelltuning_language` in `dictman`. The English frequency list for instance would thus be called in `spelltuning_en`.
threshold	The minimum frequency for terms in the frequency list. All entries that have a lower frequency will be ignored for the alignment process.

Chapter

5

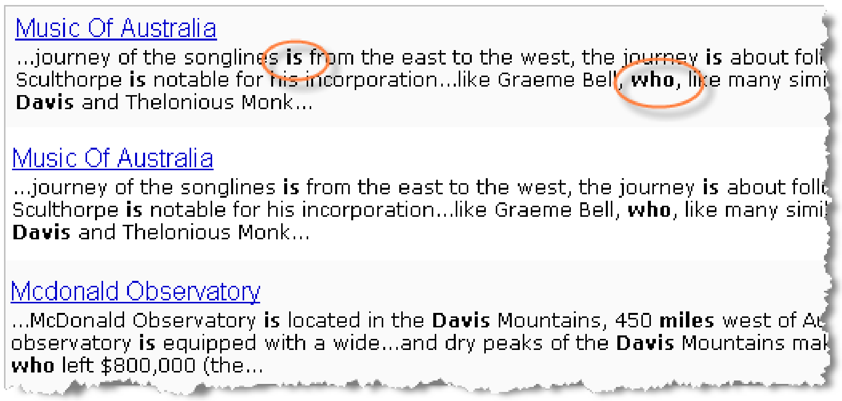
Anti-Phrasing and Stop Words

Topics:

Example: The query "Who is Miles Davis?" is reduced to "Miles Davis", which will give you more precise results.

Anti-phrasing is closely related to the concept of **stopwords**. In contrast to stopwords, however, the anti-phrasing feature does not remove single words, but entire phrases. Removing single words implies the risk of removing important words that happen to be identical with stopwords. Phrases, in contrast, are more unambiguous and can therefore be removed from the query more safely. Therefore, the antiphrase dictionary shipped with ESP does not contain single words. However, if desired, single stop words can also be added to the anti-phrase dictionary using the dictionary maintance modules.

The default dictionary for applying anti-phrasing is a common dictionary for all supported languages.

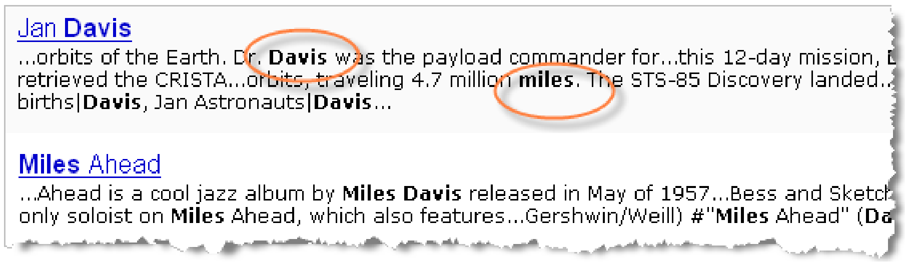


Music Of Australia
...journey of the songlines **is** from the east to the west, the journey **is** about foll
Sculthorpe **is** notable for his incorporation...like Graeme Bell, **who**, like many simi
Davis and Thelonious Monk...

Music Of Australia
...journey of the songlines **is** from the east to the west, the journey **is** about follo
Sculthorpe **is** notable for his incorporation...like Graeme Bell, **who**, like many simi
Davis and Thelonious Monk...

Mcdonald Observatory
...McDonald Observatory **is** located in the **Davis** Mountains, 450 **miles** west of Au
observatory **is** equipped with a wide...and dry peaks of the **Davis** Mountains mak
who left $800,000 (the...

Figure **4**: The query "Who is Miles Davis?" without anti-phrasing may result in hits on both "who" and "is"



Jan Davis
...orbits of the Earth. Dr. **Davis** was the payload commander for...this 12-day mission, D
retrieved the CRISTA...orbits, traveling 4.7 million **miles**. The STS-85 Discovery landed..
births|**Davis**, Jan Astronauts|**Davis**...

Miles Ahead
...Ahead is a cool jazz album by **Miles Davis** released in May of 1957...Bess and Sketch
only soloist on **Miles** Ahead, which also features...Gershwin/Weill) #"**Miles** Ahead" (Da

Figure **5**: With anti-phrasing enabled, you will not get hits on "who" and "Is"

Supported Languages for Anti-Phrasing

Dictionary entries for anti-phrasing are included for the following languages:

Note: Anti-phrasing for languages not shipped with the product can be requested. Contact your FAST Account Manager or FAST Technical Support for details.

Shipped with product	On request	On request
• Dutch • English • French • German • Italian • Japanese • Korean • Norwegian • Portuguese • Spanish	• Arabic • Czech • Danish • Estonian • Finnish • Hebrew • Hungarian • Hindi • Latvian • Lithuanian	• Polish • Romanian • Russian • Slovak • Swedish • Turkish • Ukrainian

FAST also provides lists of simple stop words for the languages listed in the following table. They can be integrated in the anti-phrase stage using the dictionary editing tools.

Note: Single stop words lists are not shipped with the product. Contact your FAST Account Manager or FAST Technical Support.

Single stop words on request	Single stop words on request
• Dutch • English • French • German	• Italian • Norwegian • Portuguese • Spanish

Stop Words

FAST provides stop word dictionaries for some languages on request. For other languages, it is possible to do simple stop word removal by adding stop words to the default dictionary for anti-phrases, using the dictionary management tools. Inclusion of single stop words has the effect that stop words are treated in the same way as anti-phrases and are removed as well. This can result in the removal of words that you actually wanted to keep.

Removal of stop words in queries should not be confused with the removal of stop words by the `vectorizer` or `lemmatizer`. Both modules have their own stop word lists which are language specific and are stored in `$FASTSEARCH/resources/dictionaries/stopwords/`, as for example `vec_stopwords.aut` or `lem_stopwords.aut`. Those lists influence the behaviour of the `lemmatizer` and the `vectorizer` only. Because these stop word lists are language specific they are less ambiguous than stop words which are to be applied to all languages. The motivation for putting a word in the stop word list for lemmatization or vectorizatoin is totally different from the one for stop words for queries. For example, you might want to put a phrase like *miles* in the lemmatizer stop word list, because this is a frequent proper name in your documents, to avoid hits on this term for *mile*, but you would probably not add this to a query stopword (anti-phrasing) list.

Query Parameters for Anti-phrasing

Anti-phrasing be controlled by the `didyoumean` query parameters.

Anti-phrasing is influenced by the `spell` and `qtf_didyoumean:rewrite_post` query parameters. Refer to the the chapter on spell checking (*Query Parameters for the Spell Checking Framework* on page 57) for more information.

Dictionaries for Anti-Phrasing

The anti-phrasing stage uses a common dictionary for all supported languages. The compiled dictionary is located at `$FASTSEARCH/resources/dictionaries/stopwords/indep_antiphrases.aut`. It is possible to create and edit anti-phrasing dictionaries with the dictionary management tools.

Chapter

6

Synonyms

Topics:

Synonymy is a powerful tool to enable search across word forms or sequences which express the same or related underlying concepts. The synonym engines can be used to implement everything from simple word to word correpondences to complete ontologies.

The synonym modules in FAST ESP do not only handle true synonyms (e.g. *seagull:gull*, *laptop:notebook*), but also spelling variants (e.g. *normalization, normalisation*) and acronyms (e.g. *BBC, British Broadcasting Company*). It can also be deployed to implement terminology support or cross-language search, provided that right type of data is available.

In FAST ESP, synonyms and spelling variations can be handled at indexing or query-time.

Example

How can I make my page **My Favorite Cars** show up in search results when a user searches for the term *automobile* ?

The key is that *automobile* and *car* are synonyms. You can set up the synonym feature so that querying for either of these two words will return the same search results.

Specifically, if you want to find a document that contains the word *cars* when you search for *automobile*, then you should add the term *automobile* to the synonym list for *cars* by enabling the spelling variation functionality.

Synonym Types and Strategies

This section covers the effect on synonym expansion on the search experience and the possible strategies to achieve search with synonyms.

A challenge in synonym handling is the potential ambiguity of both input and output forms. This can lead to precision problems. Words can have several meanings that might be overlooked when setting up a synonym dictionary. Examples for ambiguities are: *mouse* (animal;device), *apple* (company name; fruit); *fast* (company name; common adjective).

The effects of synonymy treatment in a search engine depend on the type of synonym/variations you are using. The following table gives an overview:

Type	Example	Effects on recall	Effect on precision and relevancy
Spelling variants	normalization, normalisation	higher recall	Generally, retrieving spelling variants does not affect precision negatively.
Acronyms and abbreviations	BBC - British Broadcasting Company, OK - Oklahoma	higher recall; if effects on recall are extreme, effects on precision will probably be problematic	Acronyms can be highly ambiguous. Short Acronyms tend to be synonymous to words. Wrong acronym search setup can have devastating effects on precision.
True synonymy	seagull - gull; laptop - notebook	higher recall	Problematic in case of ambiguities. To be handled with care.

Synonym strategies

Synonyms can be added to the query or to the index.

- Query side synonyms: Queries can either be rewritten to contain the synonyms, or the synonyms can be issued as suggestions. This strategy allows to control synonymy on a per query basis independently of lemmatization.
- Index side synonyms: Synonyms can be added to the indexable terms during document processing. In ESP index synonyms are handled in the same ways as lemmas - i.e. they can be switched on and off at query time through the qtf_lemmatize parameter.

Synonyms dictionary entries can be handled in different ways.

- Two way synonyms: those are synonyms are expanded in both directions, e.g. *notebook-laptop* could be entered as a two way synonym. In the master dictionaries, there is a flag available to steer this behaviour.
- Rewrite synonyms: For some purposes, you might want to define synonyms that replace the original terms. This can be done in the query side synonym engine.

Query-Side Synonym Expansion

Synonyms and spelling variation expansion can be applied at query time using an instance of `QT_Synonym`

Any modification of a query is carried out by query transformers in the query transformer framework. One of them, `QT_Synonym`, analyzes a query, compares it to a pre-defined set of dictionaries containing synonymous terms, and

- modifies the query by adding supplementary terms or phrases
- generates a list of synonyms, acronyms or spelling variations, and provides this list as a set of suggestions for query improvements to the user.

Enable Query-Side Synonym Expansion

This topic describes how to enable synonym expansion at query time.

`QT synonym` is by default part of any qt-pipeline. Query-side synonyms can be configured with the following methods:

- You can create and edit synonym dictionaries using the `LinguisticsStudio` or Dictman.
- You can set and edit search profile specific synonymy within the Search Business Center.
- You can manually configure a `qt_synonym` stage by editing the QRServer configuration file.

Query-Side Synonym Dictionaries

This section provides information on how query side synonym dictionaries are deployed to ESP.

Query side synonym dictionaries should be placed in the `$FASTSEARCH/resources/dictionaries/synonyms/qt` folder, and the instance of `QT_Synonym` in `$FASTSEARCH/etc/config_data/QRServer/webcluster/etc/qrserver/qtf-config.xml` should be updated with the name of the dictionary.

Note: Changes will not take effect until you run the following command: `$FASTSEARCH/bin/view-admin -m refresh` This will ensure that the changes take effect on all QR servers at the same time.

Note: If the tool in SBC for adding synonym dictionaries to search profiles is used, dictionaries will be deployed to the system fully automatically. Thus, there is no need to edit `qtf-config.xml`

Parameters for the Query Side Synonym Framework

The behaviour of `qt_synonym` can be controlled by a number of parameters settings, both query parameters defined for each query and static parameters defined for a specific instance of the QT.

Query parameters

Those parameters control the behaviour of qt_synonym on a per query basis. If you have a new stage based on qt_synonym, use qt_YOURSTAGENAME instead of qtf_synonym: as prefix.

Query parameter	Valid values	Description	Default
qtf_synonym:querysynonyms=value	true, false	Switches on or off query side synonymy.	depends on static parameter
qtf_synonym:use_dictionaries=dict1:dict2	any dictionary filename list (file names only, no path)	Switches on the specified dictionaries	all dictionaries
qtf_synonym:suggest_with_original=value	true, false	suggestions: switches on or off inclusion of original term	depends on static parameter
qtf_lemmatize=value	true, false	Switches on or off document side synonymy together with lemmatization	false

Static parameters

Those parameters are defined in qtf-config.xml for any stage based on qt_synonym

Static parameter	Valid values	Description	Default
on_by_default	0,1	If set to 1, the stage is used by default. Can be overriden by query	0

Static parameter	Valid values	Description	Default
		parameter qtf_synonym:querysynonyms	
synonymdictN (where N is 1-20)	PATH_TO_DICTIONARY or FIELD1;FIELD2:PATH_TO_DICTIONARY	Defines a dictionary used for the specified root scopes. The fields specification can be omitted (defaults to all root scopes). Absolute paths are not possible together with field specifications.	no default value
suggest_with_original=value	0, 1	suggestions: switches on or off inclusion of original term. Overridden by query parameter.	0

Index-Side Synonym Expansion

This topic describes how synonyms and spelling variations are added to the document during document indexing.

Document side synonymy is handled in the processor `SynonymExpander` . This processor is configured for the attributes that contain the text for which synonyms are to be retrieved. Synonyms will be added as annotations to the base forms or can be written to a separate output field. The synonym processor uses the configuration file `$FASTSEARCH/etc/SynonymConfig.xml` or a custom configuration file.

Your synonym dictionary contains the entry *laptop:[notebook]* . The document you are pushing through the pipeline has the following content: *I bought a laptop in the store.* The synonym *notebook* will be added as an annotation at the same position as *laptop*.

Enable Synonym Expansion During Document Indexing

This topic describes how to enable synonym expansion during document processing.

Document-side synonyms can be configured with the following methods:

- You create and edit document-side synonym dictionaries using the `LinguisticsStudio` or Dictman.
- You can include the synonym expansion stage and configure synonyms for specific fields and document languages

In order to make a page containing the word *cars* show up in a search result when searching for *automobile*, follow this procedure.

1. Include a custom stage based on the `SynonymExpander` processor in the pipeline used to feed documents.

   The `SynonymExpander` only works properly if it is placed between `tokenizer` and `lemmatizer` and if it is configured for languages that are lemmatized, and if the fields for which it is set up support lemmatization.
2. Add the automaton reference (dictionary that contains the synonyms) to the configuration file, preconfigured as `$FASTSEARCH/etc/SynonymConfig.xml` .
3. Restart the document processor and check for errors in the log.
4. Add the term *automobile* to the synonym list for *cars* .
5. `qt_lemmatizer` must be present in the qt-pipeline used.
6. Feed documents.

Index-Side Synonym Dictionaries

Index-side compiled dictionaries have to be manually distributed from the working directory to the `$FASTSEARCH/resources/dictionaries/` folder on all nodes that have a running FAST Query and Result Server. It is recommended to place the dictionaries for document side synonym expansion in `$FASTSEARCH/resources/dictionaries/synonyms/dp` .

SynonymExpander

This document processing stage produces synonyms for selected attributes.

This stage searches the content of the elements specified by the `attributes` parameter. The processor is language sensitive and uses the *languages* attribute

Parameter	Description	Default value
attributes	This parameter specifies the fields to which synonyms are added. Field names are separated by a blank space. If a field name is in the form `source:target` , the synonyms are not added to the source field, but written into the field specified as the target field.	title headings body
configfile	This parameter specifies the synonym configuration file.	`$FASTSEARCH/etc/SynonymConfig.xml`

Interaction with other Modules and Features

Synonyms and Lemmatization

Document side synonymy: In ESP, this strategy works in conjunction with lemmatization. The use of document side synonyms can be switched on or off at query time using the query parameter qtf_lemmatize=on/off. For an optimal result, synonyms should be added to the dictionary:

- in all forms if you are using lemmatization by expansion
- in the base form if you are using lemmatization by reduction

Query side synonymy: Depending on the order of both QTs, `QT-Synonym` receives lemmatized or non-lemmatized queries and its output will be lemmatized or not. In the standard setup, QT-Synonym receives non-lemmatized content. This means that single word synonyms are automatically subject to lemmatization in case lemmatization is activated for a query. Multi word synonyms are not subject to lemmatization for lemmatization by expansion, because lemmatization does not support proximity.

Synonyms, Tokenization and Character Normalization

Synonyms and Tokenization

Document side synonymy: The SynonymExpander stage receives the content that has been processed by the tokenizer. The input is therefore lowercased and normalized according to the tokenizer configuration. Document side synonym dictionaries edited with the dictionary editing tools are automatically tokenized using the tokenizer configuration.

Query side synonymy: The synonym query processor stages receives the content that has been processed by the tokenizer query processor. The input is therefore lowercased and normalized according to the tokenizer configuration. Synonym dictionaries edited with the dictionary editing tools are automatically tokenized using the tokenizer configuration.

Note: If you use dictionaries that have not been built using the dictionary editing tools, synonym might not be handled correctly.

Note: If you change the tokenizer configuration, synonym dictionaries have to be recompiled from the sources.

Synonyms and Character Normalization

If character normalization is configured in the tokenizer configuration (the recommended method), this is handled as described in the previous section.

If you set up character normalization using the separate characternormalizer stage, you have to set the property `compWithCharNormalize` to `true` in the dictionary editing tools and recompile the synonyms dictionary.

Synonyms and Query Highlighting

This topic describes information on how query highlighting is used in conjunction with synonyms.

Document synonym highlighting is not performed by Juniper, but query side synonyms are highlighted.

Chapter

7

Entity Extraction

Topics:

Both pre-defined and customized entities shipped with FAST ESP can be detected and extracted. Extraction of pre-defined entities is supported out-of-the-box for English, German, French, Korean, Spanish, Portuguese, Japanese, Italian, Dutch and Chinese.

Examples of pre-defined entities are:

Named entities	Other entities
• person • company • location • newspaper • university • airline • car • street names	• job title • date • price • measure • acronym • e-mail • file name • ISBN • phone numbers • zip code • stock tickers • time • quotation

Entity extraction is, by default, part of the NewsSearch processing pipeline for extracting entities on the document level and the Semantic pipeline for extracting entities on the scope level. Entity extraction can, however, be used in custom document processing pipelines as well.

Extraction of other entities is possible by:

- using the Admin GUI to specify additional extractors
- via a regular expression document processor which supports entity extraction based on regular expressions. The default configuration of this document processor supports extraction of E-mail addresses and US locations. Additional regular expressions can be defined, for example, to extract product names or customer specific information.

For support on extending the entity extraction feature, contact FAST Technical Support.

Information Available with Entity Extraction

Entity extraction not only recognizes entities in documents but also enables the normalization of entities to a canonical form and addition of meta information to extracted entities.

Meta Information for Entities

For certain types of entities additional interesting information ('meta information') is provided. For locations, for example, information is provided about the type of location (whether it is a country or city; the longitude and latitude etc.). For scopes, this information is stored in form of attributes to the corresponding tag:

- Peter lives in <location country="USA" region="America" subregion="North;MA" coordinates="42.35,-71.05" base="Boston, Ma" class="city">Boston</location>

Furthermore the meta informtion is displayed in the entity navigator in the SFE

The meta information provided might not be available for all entities of a certain type due to missing information or because of ambiguity issues. If you want to add more meta information, please contact FAST Technical Support.

Normalization of Entities

Entities can often be referred to by using different variants. For example, the two forms "United Kingdom” and "UK” both refer to the same entity. To deal with such cases it is possible to enter in the dictionary the form which should be recognized in a document ('surface form') and the canonical form ('base form'). The effect is that if you use scope search the base form is stored in an attribute ('base') of the tag which is used to annotate the recognized entity, as e.g.:

- <person>Peter Smith</person> now lives in the <location base=”UK”>United Kingdom</location>. He had moved to <location base=”UK”>UK</location> in 2000.

Furthermore, the base form is used to display the recognized entities in the entity navigators so that entities will be displayed in the same way regardless of orthographic variants used to refer to the entity.

Configuring Entity Extraction in the Index Profile

The index profile is where you can indicate which entities should be written to which fields. Note that you do not have to define a field for each type of entity you want to extract.

Furthermore, you can specify in the index profile which entities you want to display in navigators and what the name of the navigators should be.

Entity Extraction During Document Processing

Entity Extration is performed by different entity extractors, i.e. document processor stages. They are all based on the document processor "Matcher", which is defined in `$FASTSEARCH/lib/python2.3/processors/linguistics/Matching.py(c)`

All of the individual stages (e.g. PersonExtractor1) can be configured in the AdminGUI when you choose "Document Processing" and then any of the processors which are listed under "Custom Stages" and which are mentioned below in the chapter "Overview of Available Entity Extractors".

All of the entity extraction stages work on the output of the FastHTMLParser and therefore have to be placed after this stage. Note that the entity extractors do not work on the output of the tokenizer because many entities are made up of idosyncratic tokens with many variations, such as "AT & T" or "AT&T".

Parameters for Entity Extractors

The following table lists all the configurable parameters for entity extractors.

Parameter	Description	Remark
input	Defines the fields from which entities are extracted.	For most extractors this is: title body xml
matcher	The `matcher` parameter names an XML configuration file that defines the type of matcher to use, and how the matcher is configured to operate. You can specify different matcher configurations for different languages (or based on any other dispatch attribute), i.e., the configurations may be provided on a per language basis, e.g., 'en:configuration.english.xml de:en:configuration.german.xml {ja,ko,zh}:configuration.asian.xml The character '*' is used to refer to any language which is not mentioned explicitly ; so could e.g. have *:configuration.fallback.xml'. Just specifying 'configuration.xml' is considered equivalent to '*:configuration.xml'.	The configuration files are in the directory: `linguistics/extractors/` Example: linguistics/extractors/configuration.personnames.en.xml
output	The `output` parameter defines the field to which the extracted entities will be written. The `output` parameter also serves as the base name for other attributes that the document will be updated with, and that contain various different views of the set of extracted entities. If the `output` parameter has the value 'X', then the document will be updated to contain attributes 'X', 'X_raw' and 'X_counts'.	Not all entities are written to output fields. For the personextractors e.g. 'output' is set to 'personnames'
filter	The `filter` parameter lists a set of types that detected matches needs to be checked against, and how "collisions" are defined for these types. If a detected match "collides" with a previously identified match of any of the listed types, then the detected match is rejected. What "collides" means is defined by filter predicates which are explained below.	For example, for 'Locationextractor1' this parameter is defined like that: airline:intersects company:intersects person:intersects university:intersects *:illegal
rename	Depending on the configuration of the underlying matcher object, matches may have extra meta data associated with them on a per match basis. The default names of these meta data fields may be overridden and assigned custom names by defining a name mapping in the 'rename' parameter. Names that are not listed are not overridden, i.e., they are assumed to map to themselves. The format of the mapping is 'X:Y', as e.g. 'meta:info', i.e. instead of the name 'meta' the name 'info' is used.	For an example where 'rename' is used, cf. LocationExtractor1 where you have e.g. 'a:country' meaning that the attribute 'a' which is set in the dictionary is renamed to 'country'. The shorter name 'a' is used in the dictionary to safe space.
byteguard	The 'byteguard' parameter enables one to specify minimum sizes (measured in bytes) of text chunks to be fed into the underlying matcher. Optionally, maximum sizes can be listed, too. Sizes can be specified on a per field basis, e.g., "body:100:1000 title:5". Fields listed in 'input' but not in 'byteguard' are assumed having a guard tuple of (0, 10000) bytes.	
guard	If specified, the stage will only be run if the field listed in the 'guard' parameter has a value.	cf. chapter 'Different types of Extractor Stages'

Parameter	Description	Remark
separator2	Depending on the underlying matcher configuration, the meta data string associated with each returned match may actually encode several individual pieces of meta data. If this is the case, the value of the 'separator2' parameter may specify a separator used to parse/split the returned meta data string into its individual components. It is assumed that the returned meta data string encodes an alternating sequence of key/value pairs, e.g., 'key1/value1/key2/value2'. In this example, the value '/' would be natural to assign to the 'separator2' parameter.	Cf. LocationExtractor1 where the the parameter is set to '/'
phrases	If the 'phrases' parameter is set to 1, then only matches that consist of at least two words will be copied out into the 'output' field.	
separator	The `separator` parameter defines the separator for different entities in the 'output' field.	;
lazy	If the 'lazy' parameter is set to 1, we do lazy loading of the matchers. That is, we don't actually load the matcher configuration until we process a document that needs the matcher.	
dispatch	The 'dispatch' parameter defines a dispatch attribute which is used to refer to the configuration file for the matcher. It is normally set to 'language' because different configuration files are used depending on the language attribute set in the document.	
type	The 'type' parameter defines the name of the semantic type of the matches, e.g., 'person' or 'location'. This name is used by the 'filter' parameter.	
meta	The 'meta' parameter defines a dictionary of meta data that all extracted matches will be annotated with.	

Filter Predicates

Let [p1, p2] denote the position offsets of the captured candidate match produced by the current stage, and let [q1, q2] denote the position offsets of a previously captured and accepted match produced by a named stage. The following "collision predicates” are currently available:

- **illegal** :The candidate match is rejected if the intervals [p1, p2] and [q1, q2] intersect but do not form an inclusion relation.
- **inside** : The candidate match is rejected if [p1, p2] is included in the interval [q1, q2]. Inclusion is reflexive.
- **inside2** : Similar to inside, but requires proper inclusion. That is, range equality is not allowed.
- **contains** : The candidate match is rejected if [q1, q2] is included in the interval [p1, p2]. Inclusion in reflexive.
- **contains2** : Similar to contains, but requires proper inclusion. That is, range equality is not allowed.
- **intersects** The candidate match is rejected if [p1, p2] intersects the interval [q1, q2].
- **before** : The candidate match is rejected if [p1, p2] comes before, i.e., lies to the left of, the interval [q1, q2].
- **after** : The candidate match is rejected if [p1, p2] comes after, i.e., lies to the right of, the interval [q1, q2].
- **equals** The candidate match is rejected if the intervals [p1, p2] and [q1, q2] are equal.

Different Types of Extractor Stages

As mentioned before, entities are extracted by different document processors stages. For each type of entity there is at least one stage in the pipeline. However, for some types you can have more than one stage extracting this type of entity. The types of entities which are not recognized by a single stage are the following:

Companies and Persons

There are 4 different stages to extract these types of entities. Let us take the company extractor as an example (the person extractor works analogously). The first two stages, "CompanyExtractorWhiteListSpecific" and "CompanyExtractorWhiteListAny" function as whitelists. They consult dictionaries which can be built up by the user. They are meant to contain entities which are not extracted by the default extractors but which should be recognized in a certain installation.

The next extractor, "CompanyExtractor1", is a stage which extracts entities based on dictionaries and general patterns. By using patterns it is e.g. possible to recognize new companies, i.e. companies which are not stored in a dictionary. There is e.g. a rule saying that a company name can consist of an uppercase word followed by company specific suffixes as e.g. "Ltd." or "Inc." so that you would recognize e.g. "Bestbet Ltd." as a company name although this string is not stored in a dictionary. Normally, there is no need to change the patterns but if you want to do so, please contact FAST Technical Support.

The last stage, CompanyExtractor2, is a partial matcher. This stage recognizes partial strings of already recognized entities. If "Bestbet Ltd." has been extracted by CompanyExtractor1 somewhere in a document and at another place in the same document you have just "Bestbet" mentioned (without the "Ltd." suffix), then this string will be recognized by CompanyExtractor2 as the same company which was recognized before, i.e. "Bestbet Ltd.". Person extractor2 works in a similar way: If a full name as e.g. "Peter Smith" is recognized by PersonExtractor1 and just "Smith" is mentioned somewhere else in the same document then this string will be recognized by the PersonExtractor2 and identified with the entity "Smith".

Locations

In contrast to the before mentioned extractors there is no partial matcher for location names because this would not make sense. The other 3 stages, however, work in the same way as described for company and person names.

Ticker Symbols

There are 3 stages to extract ticker symbols: The first stage (TickerExtractor1) extracts ticker symbols based on dictionaries and patterns. The second stage also extracts tickers based on dictionaries and patterns but that is only done if ticker symbols were already extracted by stage 1, i.e. stage 2 is only applied if stage1 has already extracted some entities. The idea behind having two different stages is that ticker symbols can be very ambiguous and that a good disambiguation strategy is to require that very ambiguous candidates for ticker symbols are only allowed to be recognized as ticker symbols if they occur in a document together with other ticker symbols which are not ambiguous. Technically this is achieved by setting the parameter "guard" in the configuration of TickerExtractor2 to 1.

TickerExtractor3 works as a partial matcher, i.e. it extracts partial forms of ticker symbols which were extracted in stage1 or stage2.

Acronyms and Teams

There are 2 stages for these types of entities, respectively. The first extracts entities whereas the second stage functions as partial matcher extracting base forms which were previously recognized in stage1.

Ordering of Stages for Entity Extractors

The order of the stages in the pipeline is important because of two reasons: First, there are some stages which work on the output of a previous stage. For example, stages consisting of partial extractors have to be called after the stage which extracts the full form of the entity for which a partial form is then extracted. Second, it is possible to define filters which can be used to make the extraction of entities in a stage depend on the presence or absence of entities extracted in earlier stages which will be described in more detail in the next chapter.

Filters for Entity Extractors

Extracted entities can be filtered in the following way: By means of the parameter "filter” it is possible to filter out entities, i.e. to prevent strings of text from being recognized as an entity that would otherwise have been extracted as such an entity. There is, for example, a filter for locations (cf. the stage "LocationExtractor1") saying that if a string of words could be recognized as a location and if the same string was recognized as a person name by a stage applied before the location extraction then the string is not to be tagged as a location. This filter prevents an interpretation of "Washington" as a location in a sentence as

- George Washington was an American president.

because "George Washington" will be recognized as a person name in PersonExtractor1 (which is located in the pipeline before LocationExtractor1) and the interpretation of "Washington" as a location will be filtered out, i.e. you will only have "George Washington" being extracted as a person and not as a location.

In a sentence like

- Peter lives in Washington

however, "Washington" is extracted as a location, because there is no person filter that would prevent "Washington" from being recognized as a location.

Tuning of Entity Extractors

There are two situations when you want to change the behavior of an extractor, i.e. when you want to change the output of an extractor: Either you want to suppress an identified entity, i.e. you do not want that certain strings (text chunks) get recognized as a certain kind of entity or you want to add an entity, i.e. you want that certain text chunks get recognized as a certain kind of entity.

To block certain entities can be done by providing a blacklist of entities which should not get recognized as a certain type of entity. You can define such a blacklist for a navigator displaying entities in the front end. To do that has the advantage that you do not have to reindex your documents to prevent certain entities from showing up in a navigator. The disadvantage is that the entities which are listed in this blacklist still get written to the index fields if such fields are defined in the index profile and they will be written to the corresponding scope fields.

For the most important entities, i.e. locations, companies and persons, it is therefore also possible to define blacklists which will be taken into account during indexing. The entries of these lists do not get recognized at all as entities of the corresponding types. Besides blacklists it is also possible to define whitelists for entity extractors so that the entries in these lists get recognized as the desired entities during document processing regardless of the predefined entity processor stages.

Black- and Whitelists for Entity Extractors

Initially these lists just contain sample entries and can be changed using `LinguisticsStudio`. For example, if you add the string "Tom Tailor" to the blacklist for person names because you know that in your content "Tom Tailor" refers to a clothing manifacturer, the text chunk Tom Tailor" will no longer be recognized as a person name. Similarly, if you add "Jim Clever" to the whitelist for companies, this text chunk will be recognized as the name of a company.

More specifically there are two types of black- and whitelists, respectively: Language specific ones (as e.g. companies_whitelist_en.txt) which are only used for the specified language, i.e. only if the language of the document corresponds to the chosen language. Besides these there are also lists (as e.g. companies_whitelist_any.txt) which are not language specific, i.e. used for any language. So if you know that "Tom Tailor" will occur in documents of any language, you would put it in companies_blacklist_any.txt and if you want "Jim Clever" only identified in Spanish documents as a company name you would put it in companies_whitelist_es.txt.

Note that there are no filters applied by default to whitelists, i.e. whatever you put in a whitelist will get recognized. As mentioned before, there is e.g. a filter for locations extracted in stage1 filtering out locations which are also identified as person names. This filter will not be applied to locations extracted by means of whitelists, i.e. if you have a string "S" in a whitelist and "S" will also be extracted as a person name you will still get "S" as a location. Furthermore, if there are contradictory entries in a white- and a blacklist (e.g. you have "X" in companies_blacklists_any.txt and in companies_whitelist_en.txt) whitelists always take precedence over blacklists (for example, "X” would get recognized in English texts as a company but not in any other language).

You can add meta information to your entries in whitelists by entering the meta information in the "value” part of the whitelist entry in `LinguisticsStudio`. For example, if you put "Country:Spain” in the value part of the entry for "Jim Clever” this entity will get tagged like that:

- <company meta=”Country:Spain”>

After having made changes to black- or whitelists you have to restart the processor server so that the new black or whitelists take effect. However, the changes will only affect documents which you feed after having made the changes, i.e. you will have to reprocess or refeed already indexed documents so that the changes will be applied to the already indexed documents as well.

Overview of Available Entity Extractors

In this chapter an overview is given of all available entity extractors, i.e. document processor stages used to extract entities.

Each document processor stage extracts a certain entity whereby it is possible that a certain type of entity is extracted by means of several different stages. The following table gives an overview over all available entity extractors. The column "Languages" indicates whether there exist different configuration files for different languages. However, even if there exists only one configuaration file for all languages it might nevertheless be the case that the phenomena which are covered are specific for a certain country (e.g. US newspaper extractor).

Name of Stage	Description	Example	Languages
AcronymExtractor1	Extracts acronyms which occur together with the definition of the acronym	<acronym definition="magnetic resonance imaging" base="mri">MRI</acronym>	independent
AirlineExtractor	Extracts airlines	`<airline base="Air France" meta="code/AF/country/France">Air France</airline>`	independent
CarExtractor	Extracts car models	`<car base="Audi A3">Audi A3</car>`	independent
CompanyExtractorWhiteListSpecific	Extracts companies which are defined in a whitelist by the user	The list is initially empty and has to be filled up by the user if needed	Language dependent
CompanyExtractorWhiteListAny	Extracts companies which are defined in a whitelist by the user	The list is initially empty and has to be filled up by the user if needed	independent
CompanyExtractor1	Extracts companies based on dictionaries and rules.	`<company base="Thebestchoice">Thebestchoice Inc.</company>`	Language dependent
CompanyExtractor2	Extracts partial forms of companies.	`<company base="Thebestchoice">Thebestchoice</company>`	Language dependent
DateExtractor	Extracts dates	`<date month="01" base="2006-01-13" day="13" year="2006">13 January 2006</date>`	independent
DegreeExtractor	Extracts degrees	`<degree base="Master of Arts" meta="MA">MA</degree>`	independent
EmailExtractor	Extracts e-mails	`<email domain="fast.no" base="john.smith@fast.no" user="john.smith">john.smith@fast.no</email>`	independent

Name of Stage	Description	Example	Languages
FilenameExtractor	Extracts filenames	`<filename base="myfile.doc" extension="doc">myfile.doc</filename>`	independent
ISBNExtractor	Extracts ISBN numbers	`<isbn publisher="4165" base="ISBN 1-4165-2373-1" group="1" title="2373">ISBN 1-4165-2373-1</isbn>`	independent
JobtitleExtractor	Extracts job titles	`<jobtitle base="chief financial officer">Chief Financial Officer</jobtitle>`	English
LocationExtractorWhiteListSpecific	Extracts locations which are defined in a whitelist by the user	The list is initially empty and has to be filled up by the user if needed	Language dependent
LocationExtractorWhiteListAny	Extracts locations which are defined in a whitelist by the user	The list is initially empty and has to be filled up by the user if needed	independent
LocationExtractor1	Extracts locations based on dictionaries and rules.	`<location country="USA" region="America" subregion="North;MA" coordinates="42.35,-71.05" base="Boston, Ma" class="city">Boston</location>`	Language dependent
MeasurementExtractor	Extracts measurements.	`<measurement base="500 kg">500 kg</measurement>`	independent
OSExtractor	Extracts operation systems.	`<os base="Linux">Linux</os>`	Language dependent
PersonExtractorWhiteListSpecific	Extracts persons which are defined in a whitelist by the user	The list is initially empty and has to be filled up by the user if needed	Language dependent
PersonExtractorWhiteListAny	Extracts persons which are defined in a whitelist by the user	The list is initially empty and has to be filled up by the user if needed	independent
PersonExtractor1	Extracts persons based on dictionaries and rules.	`<person gender="M" base="John Smith">John Smith</person>`	Language dependent
PersonExtractor2	Extracts partial forms of persons.	`<person gender="M" base="John Smith">Smith</person>`	Language dependent
PhoneNumberExtractor	Extracts phone numbers.	`<phone areacode="(973)" base="(973) 427 1229">(973)-427-1229</phone>`	independent
PriceExtractor	Extracts prices.	`<price>$155</price>`	independent
QuotationExtractor	Extracts quotations.	`<quotation>"Hello," he said "my name is Frank."</quotation>`	independent
StreetExtractor	Extracts steets.	`<street base="Kennedy Blvd">Kennedy Blvd</street>`	independent
SubstanceExtractor	Extracts substances.	`<substance base=” aminotrimethylhexyl”>aminotrimethylhexyl</substance>`	independent
TeamExtractor1	Extracts US sports teams.	`<team city="Chicago, Illinois" league="National Basketball Association (NBA)" base="Chicago Bulls" stadium="United Center" sport="basketball">Chicago Bulls</team>`	independent

Name of Stage	Description	Example	Languages
TeamExtractor2	Extracts partial forms of US sports teams.	`<team city="Chicago, Illinois" league="National Basketball Association (NBA)" base="Chicago Bulls" stadium="United Center" sport="basketball">Chicago Bulls</team>`	independent
TickerExtractor1	Extracts stock ticker symbols.	`<ticker>YHOO</ticker>`	independent
TickerExtractor2	Extracts stock ticker symbols, given that at least one "safe" ticker was previously found..	`<ticker>YHOO</ticker>`	independent
TickerExtractor3	Extracts stock ticker symbols, given previously identified tickers from other places in the document.	`<ticker>YHOO</ticker>`	independent
TimeExtractor	Extracts times	`<time base="19:00">7 pm</time>`	independent
URLExtractor	Extracts URLs.	`<url base="http://www.fastsearch.com">http://www.fastsearch.com</url>`	independent
USNewspaperExtractor	Extracts US newspapers.	`<newspaper base="New York Times" meta="NY">New York Times</newspaper>`	independent
USZipCodeExtractor	Extracts US zip codes.	`<zipcode base="02101">02101</zipcode>`	independent

Search Options and Entities

Depending on which search mode you choose in the default SFE different kinds of results are obtained.

Simple Search

When you send a query using the option 'Simple Search' you will have navigators on your result page containing all the entities which are extracted in the result documents. If you use 'Advanced Controls' and select 'Show All Fields' you will see all the fields containing entities.

Contextual Search

The behavior of 'Contextual Search' is the same as 'Simple Search' with some additional options. Whereas in 'Simple Search' you can only search for free text, i.e. your query terms can occur anywhere in the documents, in 'Contextual Search' you can select entity scopes in which your query terms have to occur. For example, if you choose 'person' and enter 'bush' as a query term only those documents are returned in which 'bush' is in the scope 'person', i.e. only documents in which 'bush' is part of an extracted person name.

There are the following additional options when you have selected "Contextual Seach":

- If you select "Display markup” the teaser is annotated with all entities occurring in it.
- If you select "Align contextual navigators to highlighted terms” the entities in the navigator of the result page will be aligned with the highlighted terms.
- If you click on "FQL” after having sent a query you will see the FQL query into which your last query was translated. So this is a good means to formulate a FQL query if you are unfamiliar with the FQL syntax.

Stand Alone Tool for Entity Extraction

If you want to quickly check a certain configuration or test the output of an entity extractor you should use the standalone tool 'matcher' in `$FASTSEARCH/bin/`. For example, if you want to check whether 'John Smith' is recognized as a person, use 'matcher' like that:

- Change to the directory `$FASTSEARCH/`
- Call `$FASTSEARCH/bin/matcher -m $FASTSEARCH/etc/config_data/DocumentProcessor/linguistics/extractors/configuration.personextractor.pass1.en.xml`
- If you then enter 'John Smith was a famous person' you will get the following output:
- `(0, 10 ['John Smith'], 'John Smith', 255, 0, 'gender/M' [1], '' [0], 0, 1)`

Note that due to the fact that this is a standalone tool there might be differences in how the tool behaves compared to entity extraction in an ESP installation:

- You can only use configurations which do not depend on the outcome of other configurations. For example, it does not make sense to call a configuration for a partial matcher, as e.g. `configuration.personextractor.pass2.en.xml`, because the results of `configuration.personextractor.pass1.en.xml` on which the partial matcher depends are not available.
- 'filter' predicates have no effect so that it is possible that an entity is recognized which would get filtered out in an ESP installation

Chapter

8

Noun Phrase Extraction

Topics:

Noun phrase extraction extracts noun phrases for use in the vectorizer. Extracted noun phrases are an enhanced version of document vectors and can be used for drill-down and result clustering.

Grammatically, an NP (noun phrase) is the subject or object of a verb; it is the thing that a sentence `talks about'. Here is an example input sentence with the extracted noun phrases highlighted in *italics*:

Understanding these *concepts* makes it easier to understand how *relevancy with respect to linguistics* works in *FAST ESP*.

When looking at a sentence or an entire document, these NPs make up most of the "meaning" or "content" of it; they are the terms that would most likely be used as index terms or as search terms by a user. Therefore, these terms are also treated specially during ESP document processing: The NP extraction processors use a built-in set of rules (a "grammar") to recognize and extract these terms so that they can be presented e.g. in a "concepts" navigator or as terms to be used when searching for similar documents.

Some of these terms can still be very generic so that they don't contribute anything to the real information of a document, such as phrases like "for a good reason" or "with respect to"; these can be blacklisted so that they will not be extracted. ESP comes with a pre-defined blacklist that contains the most frequent irrelevant terms, and users can add their own terms to that list.

Conversely, there might be terms that are not matched by the general purpose rules that are used by the ESP document processors but that are still very relevant to your specific application domain, e.g. the names of certain products. These terms can be added to a special "white list" of terms that will be extracted in any case and regardless of the built-in extraction rules.

The NP extraction processors require that the part-of-speech tagger is run beforehand so that words have already been recognized as nouns, verbs etc. It is important that the tokenizer is also run prior to NP extraction in the pipeline since it creates the special document graph structure that the NP extraction processors use.

Supported Languages

Currently, the ESP standard installation contains grammars for English and German. Norwegian and Swedish are available on demand.

Document Processors

The document processing pipeline "CustomLingustics" contains an example setup of the processors needed for NP extraction. The following section list all document processors that are relevant for the task.

npTagger

This processor runs the grammar(s) for the various languages, including the grammars for blacklisting and whitelisting. The terms that have been matched by these grammars are inserted into a special document graph structure; they are copied into the position table by the "*Extractor" processors in the later stages of the pipeline.

Parameter	Description	Default value
configfile	The value of the "configfile" parameter is an information extraction pipeline configuration file which contains references to the top-level grammar file(s) to be used for the various languages, including the grammars for blacklisting and whitelisting. Note that there is no resource service or config server support for this configuration. Please contact professional services if you need to customize the behavior of the grammars.	See documentation for processor in Admin GUI
input	A space separated list of document fields that should be processed by the tagger.	See documentation for processor in Admin GUI

npStopWordsExtractor

This processor copies the stop words into the position table. The value of the "output" parameter is the name of the field into which the stop words are copied. The other parameters, such as "byteguard", "input", are the generic matcher parameters; please see the documentation of the matchers for more information on these.

npStopPhrasesExtractor

Similar to npStopWordsExtractor, but for stop phrases.

npExtractor

This processor copies the NPs into the position table. At the same time, it filters out:

- all NPs containing stop words that were matched by the stop words grammar;
- all NPs equal with stop phrases that were matched by the stop phrases grammar.

Apart from the generic matcher parameters, the important parameters for this processor are:

Parameter	Description	Default value
filter	Implements the stop word and stop phrase filtering.	See documentation for processor in Admin GUI
output	Specifies the name of the key under which the NPs are copied into the position table.	See documentation for processor in Admin GUI

EntityVectorizer

This processor creates a document vector using the NPs extracted by the npExtractor stage. By default, these NPs are merged with the terms extracted by the uppercase extractor with equal weights; this is determined by the "input" parameter:

Parameter	Description	Default value
input	A space separated list of document fields from which the document vector is composed.	See documentation for processor in Admin GUI

Part-of-Speech Tagging

The "POSTagger" document processor adds part-of-speech tags (such as "noun", "verb") to the words in the input.

POS tagging is required for subsequent processes such as the NP Extraction.

It requires that the sentence boundary detector is run previously in the pipeline. Here is an example of an input sentence and the associated part-of-speech tags produced by the tagger:

```
This    is    a      bold    statement.
DET     V     DET    ADJ     N
```

Configuration Options

The following table lists the configuration parameters available for the part-of-speech tagger.

Parameter	Description	Default value
configfile	This file contains the tagger settings for various languages. Note that there is no resource service or config server support for this configuration. Please contact professional services if you need to customize the behavior of the tagger.	See documentation for processor in Admin GUI
input	A space separated list of document fields that should be processed by the tagger.	See documentation for processor in Admin GUI

Blacklisting and Whitelisting

As described above, there is a list of terms that should not be considered relevant noun phrases, and a list of terms that will be extracted even if they would not be matched by the rules. These lists, also called "blacklist" and "whitelist", are language specific (i.e. there are separate lists for each language). They are compiled into ESPs automata and located in:

`$FASTSEARCH/resources/dictionaries/matching/npextraction_blacklist_LANG.aut`

`$FASTSEARCH/resources/dictionaries/matching/npextraction_whitelist_LANG.aut`

(where `LANG` is a 2-character ISO language code, e.g. "en" for English). They can be edited in `LinguisticsStudio`, i.e. terms can be added or removed. Note that all entries must fully match the terms to be excluded or included, i.e. no partial matching is done on the entries. So if for example you want to prevent extraction of the term "related topics", remember to also add the singular form "related topic" to the blacklist.

Please see the documentation on `LinguisticsStudio` for more details on how to edit these term lists.

Chapter 9

Vectorizer

Topics:

Vectorization is the process of computing document vectors. A document vector is a representation of the unstructured textual content that is associated with a document. In most document processing pipelines in FAST ESP there is a document processor performing vectorization. This document processor is referred to as the `Vectorizer`.

Document vectors are used for similarity search (result clustering) and to extract key words from documents. An enhanced version of vectorization is noun phrase extraction.

Configuring Stopwords for Vectorization

The vectorization system in FAST ESP has an optional configuration file (`$FASTSEARCH/etc/config_data/DocumentProcessor/linguistics/vectorizer/configuration.stopwords.xml`) that specifies the stopword lists on a per language basis. A stopword is a word that we explicitly ignore in the vectorization process.

Note: Not all vectorizer types make use of stopword lists, in which case the stopword list is simply ignored.

For English the set of stopwords might be {the, a, in, . . .}. In addition to using a dictionary we might impose stopword rules, for example that all words that contain digits are to be treated as stopwords.

Table **1**: Stopword configuration parameters
Parameters used to configure stopword treatment in the vectorization process.

Name	Value	Description
language		Specifies the language that this set of stopword filters is appropriate for. If no language is specified the filters will be used as a "fallback”. For English the language attribute would have the value *en* . Assume that there are (stopwords) tags for *en* , *de* and *fr* , plus a "fallback” tag. The "fallback” tag would be used to vectorize documents in all other languages than English, German or French. In FAST ESP the set of valid values for the language attribute is specified by ISO codes.
encoding		Specifies the character encoding scheme that this set of stopword filters are appropriate for. If no encoding scheme is specified the filters are assumed to be applicable for all encodings. The encoding attribute can be omitted in the context of FAST ESP since all text is converted to UTF-8 before being passed to the `Vectorizer` .
filename		Specifies the name of the file containing the dictionary of stopwords. This is a simple text file with one entry per line. Stopword dictionaries should be encoded in UTF-8. Generally, they should be encoded in the scheme defined by the value of the encoding attribute.
alpha		Specifies filtering rules based on the system isalpha/1 predicate.
	none	Do not do any filtering based on isalpha/1 considerations. For backwards compatibility, no is a synonym for none.
	first	A token is a stopword if its first character fails on isalpha/1. The tokens *hello* and *hønefoss* would pass this filter, whereas the tokens *ørken* and *1foo2* would fail.
	any	A token is a stopword if any of its characters fail isalpha/1 and is not in {'-', ' '}. The tokens *hello* and *billy-bob* would pass this filter, whereas the tokens *hønefoss* , *ørken* and *1foo2* would fail.
	all	A token is a stopword if all of its characters fail isalpha/1. The tokens *hello* , *billy-bob* , *hønefoss* , *ørken* and *1foo2* would pass this filter, whereas the tokens *æøå* , *4523* and *100-4* would fail.
digit		Specifies filtering rules based on the system isdigit/1 predicate.
	none	Do not do any filtering based on isdigit/1 considerations.
	first	A token is a stopword if its first character passes isdigit/1. The tokens *hello* and *john316* would pass this filter, whereas the tokens *1foo2* and *123* would fail.
	any	A token is a stopword if any of its characters pass isdigit/1. The token *hello* would pass this filter, whereas the tokens *john316* , *1foo2* and *123* would fail.

Name	Value	Description
	all	A token is a stopword if all of its characters pass isdigit/1 or is in {'-', '/', ' '}. The tokens *hello* , *john316* and *1foo2* would pass this filter, whereas the tokens *123* , *100-4* and *342/23* would fail.
length		All tokens shorter than this value will be treated as stopwords. String length is currently measured in bytes. Some languages employ multi-byte characters. In this case, the logical number of characters in a token may differ from the tokens' length in bytes.

Chapter 10

Structural Analysis (STAN)

Topics:

Structural Analysis (STAN) is a tool which is used either for text classification, or extraction of text from HTML documents. Unlike Entity Extraction, which aims at extracting smaller entities based on regular expressions, STAN extracts larger pieces of text. STAN splits the document into text chunks, and based on cues such as font size, what kind of keywords the chunk contains, and where the chunk is located on the page (top, bottom, left, right), we can define whether or not we want to extract this chunk.

In addition to extracting text chunks from the HTML page, STAN is also used to classify the contents of the page. This strategy relies mostly on keywords. A page containing words like "buy", "order" and "shopping basket", for example, is likely to be recognized as a online shopping portal.

Supported Languages for STAN

`STAN` grammars are available for the following languages:

Shipped with product	On request
• Dutch • English • French • German • Italian • Japanese (news only) • Portuguese • Spanish	• Norwegian

Basic Configuration

STAN is not aware of the document language. This means that one needs to create different processor stages for different languages.

In the ESP default configuration, there are two pipelines using STAN:

- The NewsSearch pipeline uses the Stan3News processor to extract title, author, date and text from news articles; additionally it uses Stan3NewsAttributeCopy to copy the two attributes stantitle and stanbody into the default fields title and body. By default, this processor uses the STAN configuration file news/en/rules.xml, i.e. it is configured for the English language. The language can be changed by replacing this file with one of the files contained in the directory `$FASTSEARCH/etc/config_data/DocumentProcessor/linguistics/stan/news/LANG` where `LANG` is one of the 2-character language codes listed above.
- The CustomLinguistics pipeline uses the Stan3Address processor to extract addresses from the document. Addresses consist of company or person names and optionally one of:
  - Street, City, Country
  - Telephone/Fax number
  - Email address
  - URLs

  This processor can be configured for different languages similar to Stan3News above.

Configuration Options

The STAN document processor can be configured with the following parameters:

Parameter	Description	Default value
Configuration	Holds the path to the configuration file.	See documentation for processor in Admin GUI
InputField	This parameter specifies a space separated list of attributes. The contents of these attributes are passed to STAN.	See documentation for processor in Admin GUI

Parameter	Description	Default value
MaxDocumentSize	This parameter specifies the maximum size of the input documents; documents larger than this are skipped by STAN.	See documentation for processor in Admin GUI

Other Stan Grammars Delivered with ESP

In addition to the STAN configurations described in the previous section, the directory `$FASTSEARCH/etc/config_data/DocumentProcessor/linguistics/stan/` contains configuration files for other types of text classification and extraction tasks. These are not used in the default pipeline configurations, but can be included by defining custom processors using these configuration files. They are described in the following sections.

Active In Business: "aib"

These are grammars for classifying HTML pages as 'active in business', i.e. pages of online shops or company web pages. This grammar sets the document field "standoctype" to either "onlineshop" (shops where you can buy products online) or "business" (company web pages with product information, contacts, office locations etc.).

Glossary-like documents: "glossary"

This grammar detects glossary-like HTML documents, i.e. documents containing lists of terms along with their explanations, and sets the document field "standoctype" to "glossary".

Text of a document without navigation, ads etc.: "realcontent"

This grammar extracts the truly 'informative' content on a web page while omitting page navigation elements, advertisements, copyright information, disclaimers etc. (This is similar to the news body extraction done by the Stan3News processor). This grammar sets the document field "stanrealcontent" to the extracted text.

Other extractors or classifiers

If other extractors or classifiers are required, please contact Fast Technical Support.

Chapter 11

Phonetic Search

Topics:

- *PhoneticNormalizer Document Processor*
- *Enable Phonetic Search*

It is difficult with certain words to remember the correct spelling, even when one remembers the pronunciation. This happens often with names. The `Phonetic Search` feature makes it possible to misspell a word and still get the appropriate documents returned. The feature is especially useful and powerful when documents contain names or other words that are prone to spelling mistakes.

Note: To prevent a large index, phonetics should be enabled on fields containing a small amount of text, such as the fields containing the proper names extracted by entity extraction.

To achieve search across spelling variants, both document and query terms are processed by the phonetic normalizer.

It is possible to set up different levels of normalization in one pipeline, and rank results according to the degree of phonetic normalization.

Example

You want to search for documents containing the name of the actor *Schwarzenegger* . However, you misspell the name *Shwarsenegger*.

When the document containing the name *Schwarzenegger* is indexed, it is run through the phonetic normalizer stage. A phonetic representation of the name is created and stored in a separate field. When the misspelled query is sent to FAST ESP, the query is also normalized by a phonetic normalizer. The query is then compared to the content of the field holding the phonetic representation. If there is a close enough match, the document is returned.

Phonetic search configurations are available for Dutch, English, French, German, Italian, Norwegian, Portuguese and Swedish.

PhoneticNormalizer Document Processor

The `PhoneticNormalizer` is the processor that is used for phonetic normalization on document side. It is used in conjunction with the `phoneticnormalizer` on query side.

The `PhoneticNormalizer` transforms the content of the fields it is applied to into a phonetically normalized representation, according to the rules defined in its configuration file. The idea is to map content to a canonical form which only preserves the most important phonetic information (such as the consonants and core vowels). The configuration can be adjusted to the needs of the language it is intended to be used on. A classical example of phonetic normalization is Soundex for English, others include Koelner Phonetics and Wiener Phonetics (both variants for the German language).

Parameter	Description	Default value
configuration	Holds the paths to the configuration files. The paths are entered as space-separated tuples, where each tuple consists of four items: language, path to the configuration file, source document field, target document field (these elements are separated by colon).	No Default value

Enable Phonetic Search

This task describes how to enable phonetic search.

The CustomLinguistics pipeline on the document side and the customlinguistics pipeline on the query side contain a setup for phonetic normalization for English, which can be copied to other pipelines or used as a template for setting up your own normalization.

Phonetic search is by default supported for English first names only. The configuration files in `$FASTSEARCH/etc/phonetic/` are used in both document and query processing.

Note: To configure phonetic search for other languages, contact FAST Solution Services.

1. Setup a pipeline that includes the `PhoneticNormalizer` stage.

   The `PhoneticNormalizer` must be placed after the PersonExtractor1 and PersonExtractor2 stages which extract the proper names for English.
2. Configure the `PhoneticNormalizer` to fit your needs.
3. Create a collection using that pipeline.
4. Feed documents through the pipeline.
5. Open `$FASTSEARCH/etc/config_data/QRServer/yourcluster/etc/qrserver/qtf-config.xml`

   If you are setting up this feature on a multinode installation, make sure you edit the `qtf-config.xml` file on the configserver.
6. Include the `phoneticnormalizer` instance after the `tokenize` instance in your query pipeline.
7. Make sure your query is directed to the query pipeline including the `phoneticnormalizer` instance.
8. Refresh search views `$FASTSEARCH/bin/view-admin -m refresh`

Chapter 12

Offensive Content Filter

Topics:

The filter is implemented as a separate document processor that can be added to an ESP pipeline. There is currently no equivalent processor for the query pipeline.

Document content that is run through the filter is compared to pre-defined terms in dictionaries. By using dictionaries it is easy to add, remove and rephrase terms to tune the filter as desired.

The filter can process HTML, XML and plain text documents. If you have content in some other format, it must be converted to one of the supported formats. The output of the filter is an overall score that provides an indication of the likeliness that a document is offensive.

Note: The offensive content filter does not use site information and does not take visual information (pictures) into account. This limits its functionality to pages that contain textual offensive material. For such pages it has a very high identification rate.

Many text based offensive content filters only identify single words, while the FAST ESP offensive content filter also uses multiword expressions. This allows for a closer investigation of the context.

Supported Languages for Offensive Content Filtering

The following languages are included by default in the `OffensiveContentFilter` dictionary:

Note: Additional terms and languages can be added using the dictionary editing tools *Dictionaries* on page 141.

Shipped with product	Shipped with product	Shipped with product
• Arabic • Chinese • Czech • English • Finnish • French	• German • Hindi • Italian • Japanese • Korean • Lithuanian	• Norwegian • Russian • Spanish • Swedish • Turkish

OffensiveContentFilter document processor

This processor tags or filters documents according to their offensive content score.

The processor can be used in any pipeline and should operate on content that has been tokenized. So it should be placed after the `tokenizer` processor.

Note: The score calculation is not configurable. It is not possible to make the processor assign a score larger than zero to documents that have a very low percentage of offensive words.

Parameter	Description	Default value
inputattributes	This parameter specifies a space separated list of attributes. The contents of these attributes are scanned for offensive content.	title body
outputattribute	An offensive score is calculated and stored in the element specified by this parameter. Nothing is written to the element if the document is excluded.	ocf_score
autfile	The path to the dictionary used by this document processing stage is specified by this parameter.	`$FASTSEARCH/resources/dictionaries/util/langid/ocf.aut`
dropthreshold	If *dropthreshold* is > 0, the document will be excluded if its score exceeds the threshold. A typical value for this attribute is '30'. If *dropthreshold* is set to '0', the document will not be removed, but will still receive a score	30

Parameter	Description	Default value
	written to the field defined in `outputattribute` .	
dropsilently	The *dropsilently* boolean parameter should be set to '0' to get a callback message for dropped documents or '1' to drop documents silently (only used if *dropthreshold* is greater than '0')	0

The following table can be used as a reference when deciding on what threshold levels to use.

Score value	Description
0	No offensive content identified.
0 - 20	Weak indication of offensiveness.
20 - 50	Strong indications for offensiveness.
50 ->	Certainly offensive.

Enable Offensive Content Filtering

Offensive content filtering can be performed by adding a document processing stage to the document processing pipeline.

1. Make required adjustments to the configuration of the `OffensiveContentFilter` processor using the Admin GUI.
2. Add the `OffensiveContentFilter` document processing stage to a pipeline of your choice.

   Remember to place it after the `tokenizer` stage.
3. Make sure a collection is using the altered pipeline or create a new collection.
4. Feed documents through the pipeline.

Chapter 13

Processing Content in Asian Languages

Topics:

Certain languages require specialized linguistic processing so that the content can be indexed and searched for. Chinese, Japanese, Korean and Thai are among such languages. In FAST ESP both the documents to be indexed and queries undergo linguistic processing where much of the processing is language-specific.

ESP can identify the language of each document automatically when it is sent for indexing. However as many languages share the same or similar scripts, search queries are generally too short for accurate language identification to be guaranteed. Therefore it is important that the language of a query is set by default or specified when each query is submitted. Failure to do this may result in the query being incorrectly processed.

Linguistic processing of documents and queries allows for an increase in relevance of the documents retrieved. Different settings may also be applied on the indexing side and for each query submitted. FAST ESP performs the following linguistic processing for Asian languages:

- Language and encoding detection of documents to be indexed. If a document contains text blocks which are in more than one language, the language of each block can be automatically identified.
- Asian texts are split up into words using language-specific tokenizers. Substring/N-gram based search is also supported for applications where recall is more important than precision.
- Lemmatization. Since Japanese and Korean words contain various grammatical endings, lemmatization is especially useful for processing these languages.

Introduction to Asian language processing

Short overview of Asian language processing in FAST ESP

Supported encodings

Japanese, Chinese, Korean and Thai characters can each be represented using different encoding systems in the documents to be indexed:

- Japanese: Extended Unix Code (also known as EUC-JP), Shift JIS, ISO-2022-JP and UTF-8.
- Chinese: Big5 (Traditional Chinese), GB18030 (Simplified Chinese) and UTF-8.
- Korean: EUC-KR, KSC-5601 and UTF-8.
- Thai: Windows-874 (also known as TIS-620) and UTF-8.

The Asian characters in these encodings are represented using multiple bytes. The encoding detection module automatically identifies the encoding of a document.

Basic Difference Between the Asian Languages

This section explains the basic difference between the Asian languages.

Words in Chinese and Japanese are not separated by spaces. Whiles spaces are used in Korean, many compound words do not contain spaces separating the component words. In Thai, spaces can, but need not, be used to segment sentences, phrases or names. Hence tokenization is used for all four languages to split a text into individual words.

Alphanumeric characters are used in all four languages; however Chinese and Japanese also make use of full-width Roman letters, Arabic numbers and punctuation marks. The full-width letters and numbers are normalized automatically into their half-width equivalents.

Number of alphabets used (apart from alphanumeric characters):

Chinese

Pure Chinese text contains only one writing system, namely that composed of Chinese characters. Out of the over 5,000 commonly used characters in Chinese, the Mainland Chinese government simplified 2,236 of these. They were also later adopted in Singapore. Hence a portion of the characters found in Mainland Chinese and Singaporean texts are different from those found in (traditional) Taiwanese and Hong Kong Chinese texts. A mixture of character types is generally not found in the same document. It makes no difference if the query language is set to Simplified or Traditional Chinese, as they are processed by the same tokenizer. The synonym dictionaries and entity extractors are, however, specific to each writing system.

A synonym dictionary allowing search queries to be automatically expanded into both writing sets is delivered with ESP 5.1.

Japanese

Japanese makes use of three different writing systems (in addition to Latin characters): Hiragana, Katakana and Kanji (Chinese characters). There are no strict orthographic rules about which script to use for which word; therefore many words exist in a great number of varieties, which is ideally reflected by making use of domain-specific spelling variation dictionaries. Those are not included in the standard distribution of ESP, but FAST can assist in obtaining, adapting and integrating them.

Korean

Korean makes use of two scripts, i.e. the Hangeul phonetic alphabet and Chinese characters, known as Hanja. Hanja words can be searched for using their Hangeul equivalents.

Note: Chinese characters are rarely used in Modern Korean texts and are almost never used in search queries.

Thai

Thai makes use of its own alphabet which is derived from the old Khmer alphabet. It has 28 vowels and 44 consonants.

Tokenization for Asian Languages

The operation of identifying what constitutes a word can be quite complicated. Some Asian languages, such as Chinese, Japanese, Thai and to some extent Korean, do not use space as a word separator. These languages require specialized and more sophisticated tokenizers that use more complicated rule sets as well as dictionaries.

FAST ESP comes with one default and four specialized tokenizers. The specialized ones are for the following Asian languages: Traditional and Simplified Chinese, Japanese, Thai and Korean.

For non-scope fields, tokenization of incoming Asian documents is provided by the `Tokenizer` document processor. For Japanese and Korean documents, the `Tokenizer` document processor also provides lemmatization.

For scope fields, tokenization is provided by the `ScopeTokenizer` document processor.

Note: Non-linguistic tokenization (sub-string search) is not supported for scope fields.

Note: FAST ESP fully supports the CJK related character normalizations, like mapping full-width latin characters to their half-width equivalent.

Tokenization Dictionaries

Since the specialized tokenizers rely on rules and dictionaries, there will always be some words that do not get tokenized in the desired way. This is particularly true for names, new words, and narrow industry terms. To enable the specialized tokenizers to identify such words according to client requirements, each specialized tokenizer has a custom dictionary to which the client can add terms of their own.

The $FASTSEARCH/etc/tokenizer/userdicts directory contains four empty user dictionary files with instructions on how to add new words to the tokenizer dictionaries. The Korean dictionary is the exception; that file refers to yet another file in which the words are to be added.

Enable Advanced Tokenization in Legacy Query Pipeline

If queries are sent through the legacy query pipeline, *fds4* , and require advanced tokenization, advanced tokenization must be enabled for this query pipeline first.

1. Open `$FASTSEARCH/etc/config_data/QRserver/webcluster/etc/qrserver/qtf-config.xml` .

2. Make sure the `parse` pipeline includes the `phrasetokenization` parameter.

```
<instance name="parse" type="internal" resource="FastQT_Parse">
 <parameter-list>
  <parameter name="indexmapfile" value="etc/qrserver/parseindex.map"/>
  <parameter name="phrasetokenization" value="true"/>
 </parameter-list>
</instance>
```

Disable Implicit Phrasing of CJK Queries

CJK queries that contain multiple words, but that are not separated by spaces, are treated by default as phrase queries. Follow this procedure to have such queries treated as *AND* queries instead.

1. Open `$FASTSEARCH/etc/config_data/QRserver/webcluster/etc/qrserver/qtf-config.xml` .
2. Change `name="stringmode" value="PHRASE"` to `name="stringmode" value="AND"` in the `tokenize` query pipeline.

```
<instance name="tokenize" type="internal" resource="FastQT_Tokenize">
 <parameter-list>
  <parameter name="stringmode" value="AND"/>
  <parameter name="rootfieldprops" value="etc/qrserver/parseindex.map"/>
  <parameter name="indexspec" value=""/>
 </parameter-list>
</instance>
```

Automatic Conversion Between Simplified and Traditional Chinese

If your collection contains documents in both Simplified and Traditional Chinese, then a query written in Simplified Chinese may not find matching documents written in Tradtional Chinese, and the other way around. It is recommended that you enable an automatic query conversion between the two language formats. Simplified or Traditional Chinese variants of your query will then be added automatically.

Enable Conversion Between Simplified and Traditional Chinese

How to add simplified or traditional Chinese variants of queries when the query string is written in one or the other variant of Chinese.

1. Add `szh_to_tzh.aut` and `tzh_to_szh.aut` to the pipeline you use, in the pipeline specification in `$FASTSEARCH/etc/config_data/QRServer/ webcluster/etc/qrserver/qtf-config.xml`

   Note: If a word, such as a person's name or company name, is not in the above synonym dictionary, it may not be converted. If there are certain words you would like to convert but are not in the aforementioned dictionaries, create your own query synonym dictionary containing the simplified and traditional variants of the desired terms.

Convert Numbers in Chinese Characters to Arabic Numerals

This topic describes how to convert numbers in chinese characters to arabic numerals.

When documents are fed, ESP can automatically convert all numbers written in Chinese characters to their equivalent in Arabic numerals. For example:

八万九千四百五十 to 89450

三四七八 to 3478

1. Open `$FASTSEARCH/etc/tokenizer/zh-simp.options` and `$FASTSEARCH/etc/tokenizer/zh-trad.options`
2. Set `NormalizeResultToken` to *TRUE*

Note: Some words and place names in Chinese contain numbers. If number conversion is active and the word containing a number is not in the system dictionary, the number in the word may be converted to Arabic numerals. For example, the place name 东四十条 is converted to 10 . To prevent this from happening, add the word to the tokenizer user dictionary.

Comparing Tokenization of Document and Query String

This topic describes how to compare document and query tokenization to locate any issues.

To check if your query term and document phrase are tokenized differently:

1. Create a simple text document in utf-8.
2. In the 1st line, write "000 [INSERT Search Query HERE] 000".
3. In the 2nd line, write "111 [INSERT document sentence HERE including 10 or so chars of context left and right] 111".
4. In the 3rd line, insert the following text to ensure that ESP can correctly identify the language as Chinese.

   If document is in Simplified Chinese: 今年34岁的黑龙江伊春市乌马河林业局职工蒋永彬在交清62元承包费后

   If document is in Traditional Chinese 今年34歲的黑龍江伊春市烏馬河林業局職工蔣永彬在交清62元承包費後
5. Feed the document and remember the name of the pipeline and the ID of the document.
6. At the command prompt type `getfixml -c [COLLECTION NAME] [DOCUMENT ID] | grep "bsrcbody"` .

Spelling Variants of Foreign Words in Korean

In addition to the standard spellchecker, ESP has a query side synonym dictionary which suggests standard spellings of foreign words in Korean, if the search query entered is a non-standard spelling. For example, if the user enters , or ('nanometre'), the standard spelling ('nanometer') is suggested.

To enable these spelling variations, include the `$FASTSEARCH/resources/dictionaries/spellcheck/ko_spelling_variation.aut` dictionary to the query pipeline you use, specified in `$FASTSEARCH/`

Lemmatization for Different Languages

This section includes information on how content is lemmatized for different languages.

Unlike for Western languages that have incoming documents lemmatized by the Lemmatizer document processor, Japanese and Korean are tokenized and lemmatized in one and the same operation by the Tokenizer document processor. The difference in implementation is due to the cost of processing Asian text. Combining the two operations in one eliminates the need for a second round of costly text parsing.

Lemmatization is not needed for Simplified or Traditional Chinese, as particle endings are separated from word stems during tokenization.

For Korean and Japanese, lemmatization is done by reducing both the words of the documents to be indexed and the query terms to their canonical forms. It is not possible to configure lemmatization for query or document expansion for those languages.

Just like for tokenization, it is required that you set the language of the query string to be submitted correctly so that lemmatization for Japanese and Korean works properly.

If you suspect that a Korean word is not being lemmatized properly and the ending is not being removed from the stem correctly you can add the word to the lemmatization dictionary. This is the file `$FASTSEARCH/lib/basis/kma/dicts/ham-usr.dic` encoded in CP949. The file header describes the necessary format of the entry.

Korean Synonym Dictionary

ESP contains a dictionary of approximately 40,000 technical words and names with their Korean transliterations.

Technical Word	**Korean Version**
naphthylamine	나프틸아민
toboggan	터보건
tyrannosaurus	티라노사우루스
Michelangelo	미켈란젤로

If a user enters an English term such as *Michelangelo*, ESP can suggest the Korean transliteration *미켈란젤로*. To install this dictionary, add the query synonym dictionary `$FASTSEARCH/resources/dictionaries/synonyms/ko_foreign_word_translation.aut` to the pipeline you use, specified in `$FASTSEARCH/etc/config_data/QRServer/your-cluster/etc/qrserver/qtf-config.xml`

Synonyms and CJK Processing

This topic provides information on how synonyms are treated when CJK documents are processed with lemmatization enabled.

CJK languages are tokenized in the tokenizer modules, both on query and document side. Japanese and Korean are also lemmatized in this module; the Chinese content is copied to the lemma field.

The lemmatized and tokenized text for CJK text is used as input for `QT-Synonym` . Non-CJK terms appear as in the original field. In theory you can also use a synonym step that adds CJK synonyms, but you have to be aware of the tokenization and lemmatization output for those languages.

Enable Chinese Number Conversion

This topic describes how to enable chinese number conversion.

1. Open `$FASTSEARCH/etc/LemmatizationConfig.xml` .
2. Add `<external_lemmatizer language="zh-simplified" description="BasisTech Chinese lemmatizer" />` after the japanese `external_lemmatizer`

Languages requiring special tokenization (CJK+)

This section defines the concept of CJK+ languages, which refers to languages that require special tokenization techniques.

Definition of CJK+ languages

A number of languages make use of writing systems in which a text is not divided up into the convenient units of meaning required for building the index of a search engine. Such units are commonly referred to as “tokens” or, in vaguer lay terms, “words”. In European languages, white space characters and punctuation marks are used to mark token boundaries. This convention is not without problems, for example compound words such as “living room” and proper nouns such as “New York” contain a space, even though both should be considered one token). Nevertheless, this practice produces acceptable initial keywords for search purposes . Such keywords which can then be further refined by applying, for example, automatic phrase detection and compound segmentation.

Chinese and Japanese differ significantly in that they do not make use of space characters, while Korean does not consistently use spaces to demarcate tokens or the words inside compounds. These three languages are usually referred to as the *CJK* group of languages, widely known for the fact that processing them requires, among other things, a special tokenization technique.

Apart from the CJK group, the only language for which ESP provides special tokenization is Thai. Nevertheless other languages may be added in the future. This open-ended set of languages for which special tokenization is needed is referred to as the *CJK+ languages* in this document.

ESP also provides a C++ SDK enabling any user to plug in custom- tokenizers for specific languages.

Software which automatically tokenizes any of the CJK+ languages is commonly referred to as a *tokenizer*, although other names used include *segmenter*, *morphological analyzer* (形態素解析 in Japanese, 형태소분석기 in Korean, 斷詞程序 and in Chinese). A tokenizer generally makes use of complex algorithms as well as dictionaries of varying complexity and so is able to do much more than merely splitting a text up into tokens: If the necessary additional information is contained in its dictionaries, a tokenizer can potentially perform:

- normalization of spelling variants (e.g. ja: 大學 → 大学, zh: 羣體 → 群体, ko: 개임 → 게임)
- lemmatization (i.e. reducing full forms to base forms ja: 難しかった → 難しい, ko: 처리하셨겠습니까 → 처리)
- part of speech tagging, e.g noun, verb, adjective labels
- advanced linguistic processing (some tokenizers include named entity extraction, sentence parsing, anaphora resolution and more)

The tokenizers used for processing Chinese, Japanese, and Korean in ESP perform lemmatization, POS tagging and, to some extent, normalization. The tokenizer for Thai is only able to split Thai texts into tokens. Note, however, that inflectional endings are considered to be tokens in Thai, so lemmatization is not needed.

The crucial importance and complexity of accurate language detection, tokenization and other processes such as lemmatization renders processing CJK+ documents vastly different from that of European languages. Hence, indexing CJK+ documents requires special configuration in all parts of the system, most notably in the document and query processing pipelines as well as in the index profile. This chapter describes in detail the configurations necessary for setting up a search application involving one or more CJK+ languages.

Language Codes

The following abbreviations are used as language codes in ESP and the current chapter:

ja	Japanese
ko	Korean
zh	Chinese, simplified or traditional
zh-simplified	Simplified Chinese 简体字: the Chinese script used in mainland China and Singapore. zh-simplified, szh and zh-cn are treated as synonymous by ESP.
zh-traditional	Traditional Chinese : the Chinese script used in Taiwan, Hong Kong and Macau zh-traditional, tzh and, zh-tw are treated as synonymous by ESP.

The same tokenizer is used to process both Simplified and Traditional Chinese in ESP, hence the distinction between the two is unimportant when selecting the query language, tokenizer language and SBC synonym dictionary language. The abbreviation “zh” or “Chinese” can simply be used. The distinction is, however, important for entity extraction, as this feature is script-specific.

CJK related configuration files

Statement about the encoding of CJK related configuration files, which has to be UTF-8.

When configuring ESP to handle various CJK+ language features, it may be necessary to modify various configuration files, some of which may include string parameters.

Important: As a general rule, all the configuration files in ESP are encoded in UTF-8 and must be saved as such after editing..

Character Normalization

Detailed information about the types of character normalization typically performed for languages belonging to the CJK group.

Character Normalization for Japanese

This section lists and describes the types of character normalization applicable to Japanese, both for queries and documents.

Half-width 半角 to full-width 全角 Katakana

→ カギ

Full-width 全角 to half-width 半角 alphanumeric characters

	Fast ESP 5.0

Numbers in Kanji to Arabic Numbers

一万五千	15000

Numbers written in Kanji (漢数字) are converted into their equivalent notation in Arabic numerals. Note that complex digit variants (大字) such as 壱, 弐 and 参 are not recognized. The conversion is activated by default and is carried out by the Japanese tokenizer. The Arabic numerals are tagged as lemma forms.

This is activated by default.

Deactivating Conversion of Kanji Numerals

1. To deactivate the conversion, add the following lines to the relevant `<basis-jma>` and `<basis-jma-reading>` sections in `$FASTSEARCH/etc/tokenization/tokenizer.xml`.

   <coordinator-options> <normalize-result-token>false</normalize-result-token> </coordinator-options>
2. Finally, restart the document processor and re-feed all documents in which numbers should undergo conversion.

Old-style Kanji variants 旧漢字、異体字 to modern Kanji 正字

ESP 5.1 is able to automatically convert old-style or uncommon Kanji variants (旧漢字、異体字) into their standard modern equivalents (正字), e.g. 髙 → 高, 學 → 学. 697 old-style Kanji characters with their modernized equivalents are defined. The conversion is not enabled by default, hence, searching for 高島屋, for example, will not retrieve documents containing 髙島屋.

Activate Kanji Normalization

1. Make a backup copy of `$FASTSEARCH/etc/tokenizer/tokenization.xml`
2. Stop all ESP servers
3. Replace the file above with `$FASTSEARCH/etc/tokenizer/tokenization_itaiji.xml`
4. Restart ESP
5. Refeed all documents

Character Normalization for Chinese

This chapter describes the types of character normalization applicable to Chinese.

Full-width 全角 to half-width 半角 Alphanumeric characters

Example: Fast ESP 5.0

This is activated by default.

Numbers in Chinese Characters to Arabic Numbers

Example: 一万五千 15000

Numbers written in characters are converted into their equivalent notation in Arabic numerals. Note that complex digit variants (大寫字) such as 壹 貳 and 参 are not recognized.

The conversion is activated by default and is carried out by the Chinese tokenizer. The Arabic numerals are tagged as lemma forms.

Deactivating the conversion of Chinese numerals

1. To deactivate the conversion, add the following lines to the relevant `<basis-cma>` and `<basis-cma-reading>` sections in `$FASTSEARCH/etc/tokenization/tokenizer.xml`.

   <coordinator-options> <normalize-result-token>false</normalize-result-token> </coordinator-options>
2. Finally, restart the document processor and re-feed all documents in which numbers should undergo conversion.

Character Variants 異體字 to Standard Characters 正體字

ESP 5.1 is able to automatically convert character variants(異體字) into their standard equivalents (正體字), e.g. 羣 → 群, 歎 → 嘆. The conversion is not enabled by default, hence searching for 感歎詞, for example, will not retrieve documents containing 感嘆詞. Before activating this feature, the decision must be made as to whether character variants are normalized to Traditional Chinese characters or to Simplified Chinese characters. If automatic traditional to simplified query conversion is activated, it makes no difference if normalization to Traditional or to Simplified is selected. If no automatic traditional to simplified conversion is activated, then the script should be selected which most users will enter queries in.

Normalize character variants to standard Chinese characters

1. In order to normalize character variants into standard characters, the line <normalization transformation="lowercase-simplified-yitizi" canonical="true" /> for Simplified Chinese normalization or alternatively the line <normalization transformation="lowercase-traditional-yitizi" canonical="true" /> should be added below the lines <tokenizer language="zh zh-cn zh-simplified zh-tw zh-traditional" compatibility="true"> <plug-in name="basis-cma" /> to the relevant tokenization mode in the file `etc/tokenizer/tokenization.xml`

Traditional Chinese 繁體字 to Simplified Chinese 簡體字 conversion

ESP 5.1 can automatically convert queries written in Traditional Chinese to Simplified Chinese and vice versa.

Example: 中國對外經濟貿易 ⊠中国对外经济贸易

There is a second option of translating Taiwanese Mandarin terms into Mainland Mandarin terms and vice versa, e.g.

Mainland Mandarin	**Taiwanese Mandarin**
软件	軟體
短信	短訊
老挝	寮國

Mainland Mandarin	Taiwanese Mandarin
阿拉伯联合酋长国	阿拉伯聯合大公國

The conversion is effected in two steps. Firstly the qt_synonym module looks up all the tokens in a query in the Traditional-Simplified Chinese synonym dictionary. If a token is found in the dictionary, then its Traditional or Simplified Chinese equivalent is added to the query. Word-based conversion offers a high degree of accuracy as many characters may be converted differently depending on the word they are used in.

There may, however, be tokens which are not found in the synonym dictionary, particularly those denoting people’s names, company names and product names. Such tokens are converted in the second step, where a character normalizer module is called and all unconverted characters undergo conversion.

Below is an example of the conversion of the query 軟體 金山 詞霸 meaning “software Kingsoft Powerword”. The Mainland term for “software”, 软件 is different from the Taiwanese term and . “Powerword” is a product name, which is not in the synonym dictionary.

With Mainland-Taiwanese conversion
Original Query: 軟體 金山 詞霸
After qt_synonym: 軟體 软件 金山 詞霸
After character normalization: 軟體 软件 金山 詞霸 词霸

Without Mainland-Taiwanese conversion
Original Query: 軟體 金山 詞霸
After qt_synonym: 軟體 软体 金山 詞霸
After character normalization: 軟體 软体 金山 詞霸 词霸

Activate conversion between Simplified and Traditional Chinese

1. To activate Traditional and Simplified Chinese conversion, firstly decide which synonym dictionary you wish to use:

   - `tzh_szh_conversion_with_dialect_equivalents.aut`
   - `tzh_szh_conversion_without_dialect_equivalents.aut`

   Follow the steps below, which install the dictionary *with dialect equivalents*.

2. Add the following two instances to `$FASTSEARCH/etc/QRSERVER/webcluster/etc/qrserver/qtf_config.xml`:

```
<instance name="synonym_zh" type="external" resource="qt_synonym">
  <parameter-list name="qt.synonym">
    <parameter name="enable" value="1"/>
    <parameter name="on_by_default" value="1"/>
   <parameter name="synonymdict1" value="
resources/dictionaries/synonyms/qt/tzh_szh_conversion_with_dialect_equivalents.aut"/>
  </parameter-list>
</instance>

<instance name="tzh_szh_ conversion" type="external" resource="qt_characternormalizer">
<parameter-list name="qt.characternormalizer">
   <parameter name="enable" value="1"/>
   <parameter name="extendedfeedback" value="1"/>
   <parameter name="config_file" value="lib/normalizations/zh_normalization_tzh_szh.xml
 "/>
   <parameter name="on_by_default" value="1"/>
   <parameter name="process_tokens" value="1"/>
   <parameter name="include_original" value="1"/>
   <parameter name="indexmap" value="NONE content xml"/>
   </parameter-list>
</instance>
```

3. Add the following lines to the qt_pipeline for which you wish to implement the synonym suggestions. The default qt_pipeline, “scopesearch” is used in the example below.

```
<pipeline name="scopesearch" alias="esp5">
  <instance-ref name="utf8" critical="1"/>
  …
  <instance-ref name="simplefql"/>
  <instance-ref name="tokenize" critical="1"/>
  <instance-ref name="synonym_zh"/>
  <instance-ref name="tzh_szh_conversion"/>
```

4. Save `qtf_config.xml`
5. At the command prompt, type `view-admin –m refresh –a`

Tokenization Algorithm

Background knowledge about CJK tokenization.

Before describing the configuration of the external tokenizers, a few properties of the tokenization strategy employed in the default CJK tokenizers will be now mentioned. This should hopefully eliminate any potential misunderstandings regarding which documents are retrieved using which query terms.

Chinese

- People’s names containing both surname and first name are treated as one token, hence a query containing a first name only will not retrieve documents containing full names, e.g. document containing “ ” is not found if the query is “ ”
- All grammatical particles (助詞) are treated as separate tokens, e.g. 老師 的, 同學 們, 簡略 地, 生產 了 取消 過. This means that the base word can be searched for regardless of the particle attached, e.g. a query with the keyword 同學 will retrieve documents containing the word 同學們.
- Verbal and Adjectival complements ( ) are treated as separate tokens, e.g. 表達 出來, 使用 下去, 確認 得 了, 漂亮 極 了. Hence verb and adjective stems can be searched for regardless of the following complement.
- As grammatical particles and complements are treated as separate tokens by default, there is no need for a lemmatization stage.
- Compound nouns are split up into their components, so each component word is searchable, e.g. 軌道 交通 系統, 中國 書法家 協會, 半導體 變流 技術. There is therefore no need for a separate compound decomposition feature in Chinese.
- A small number of compound nouns which are well established set phrases are not broken up into their components. The component words are therefore not searchable. e.g. the compounds 發展中國家, 有線 電視, 中華人民共和國 are each considered one indivisible token. The search query 發展 will not retrieve documents containing the compound 發展中國家.

Korean

- People’s names are treated as one token, hence first names cannot be searched for in isolation, e.g. a document containing 김철수 will not be found if the query is 철수.
- Compound nouns are only decomposed, if compound decomposition is activated, e.g. 교통사고   교통 사고, 국립역사박물관   국립 역사 박물관. Compound decomposition is activated by default. To deactivate it, see the section on tokeniziation configuration (*Fine tuning the CJK tokenizers* on page 126)
- The jeok (적,的) ending is not deleted during lemmatization, e.g. 역사적으로   역사적, 전통적인   전통적. Hence searching for 역사적 will not find documents containing 역사. If you would like queries containing

a keyword without the jeok suffix to retrieve documents containing the keyword with the jeok-suffix, install the jeok synonym dictionary.

- The base form of Sino-Korean hada(하다)-verbs and adjectives is the stem without the hada suffix, e.g. 처리하다 처리, 유명하겠습니까 유명. Users should therefore not include the hada ending in search queries, or alternatively insert a space between the stem and the 하… ending 어미 in the search query, e.g. 유명 하겠습니까, 처리 하다.
- The short passive and causative 단형 피동사 사동사 suffixes of hada( ) verbs, [stem]되다 and [stem]시키다, are removed during lemmatization, e.g. 처리될 처리, 취소시키면 취소. Hence, to search for documents containing 처리될 or 처리시키면, use the search term 처리. Search queries such 처리되다 or 처리시키다 will not return any documents.
- The lemma form of non-hada verbs and adjectives is the da( )-form, e.g. 깨내봐야지요 깨내다, 싫어하는 싫어하다.
- Short passive and causative (단형 피동사 사동사) suffixes of non-hada verbs are not removed during lemmatization, and hence such forms must be searched for separately, e.g. 씹힌다 씹히다 (not씹다), 깨워라 깨우다 (not깨다).
- The honorific pre-suffix -si- (-시- 선어말어미) of verbs and adjectives is not removed during lemmatization and such verb forms must be searched for separately, e.g. 고치셨어요 고치시다 (not 고치다).
- Colour terms in the adnominal form (관형각) are treated as lemma forms and so are not lemmatized to the da-form, e.g. 검은, 하얀, 푸른 are treated as lemma forms. All other forms are lemmatized to the da-form, e.g. 하얗겠어 하얗다, 푸르던데 푸르다.

Dynamic Hit Highlighting

Specification of dynamic hit highlighting (i.e. highlighting those strings in the result view that match the query) for CJK.

Korean

A special note is made here about dynamic hit highlighting in the results page of Korean queries, as not all hits may be highlighted.

The following search terms are not highlighted in the result documents:

- Non hada-하다 verb and adjective search terms, if the document and search term are not in the lemma form (기본형). If, for example, the search term is 탐내었던 and a document contains the word 탐내던, 탐내던 will not be highlighted. Only if the both the search term and document contain the lemma form 탐내다 will the word be highlighted.
- Pronouns where the lemma form is not a substring of the inflected form, e.g. if the search query contains the word 너 and the document contains the word 네가, the word will not be highlighted.

For all other search terms he hits are hightlight.

Index Profile

This section contains information about how to set up an index profile adequate for CJK+ languages.

Defining the tokenization method for each field

Description of the semantics of various elements of the index profile, esp. those related to tokenization and substring search.

```
<field name="body" tokenize="auto" lemmatize="yes" substring="2">
  <vectorize default="5:5" alternative="{ja,ko,zh,szh,tzh}:5:0" />
</field>
```

The field definition, such as the one above, requires special attention if it refers to CJK+ content. The parameters are described in the following table:

Specifier	**Effect**
Tokenize	`auto` means the special CJK+ tokenizer will be used automatically **if the language of each document is correctly identified** . `delimiters` denotes that the special CJK+ tokenizers will **not** be used to process this field. Only punctuation and white space will be used to detect tokens. **Important:** If this is set, no lemmatization or forms of character normalization carried out by the special tokenizer will be performed .
Lemmatize	`yes` means lemmatization is turned on. Note that for CJK+ languages, lemmatization is carried out by the pipeline stage called *Tokenizer*, not by the pipeline stage called *Lemmatizer*. Lemmatization can be deactivated for Korean and Japanese by removing the line `<dictionary forms />` from the `etc/tokenizer/tokenization.xml` file. See section on tokenizer configuration(*Fine tuning the CJK tokenizers* on page 126).
Substring	If this is set to the value N, substring search is activated. This specifies that after CJK+ specific tokenization is carried out, the tokens will be further split into N-grams. This is useful if recall is much more important than precision in a given application or the content is very difficult to tokenize, for example, because it contains many rare words, new coinages, names of products, or the like. Further details are given below.
vectorize.default	This defines two integer weights which are applied to the terms extracted from the current field to produce the document vector. Terms will be extracted from the non-tokenized original content, which, in the case of CJK+ languages, yields character sequences which may not be entirely correct.
vectorize.alternative	This defines two integer weights, applied to terms extracted **from the tokenized version** of the current field to give the document vector. This is relevant for CJK+ languages. The languages this applies to are listed in the curly brackets. If you add a special tokenizer for a new language (using the tokenizer SDK), you should consider adding that language to the list defined here.

Substring Search

This feature, often referred to as N-gram search, is normally applied to fields which are deemed hard to tokenize automatically. One reason for this is that they may contain rare words, new words such as product names or other types of strings, which are unlikely to be in the tokenizer system dictionary.

The feature should also be used when recall (i.e. the overall number of documents retrieved) is considered much more important than precision (i.e. high relevancy of the results). Without substring search activated, a CJK+ query may, in certain cases, be tokenized incorrectly and therefore return a meager or empty result list. This will never occur when substring search is used, as all N-gram substrings of each token will be indexed, as well as N-grams spanning token boundaries. For example, the string

> jp: 日本語は難しくない (“Japanese is not difficult.”) zh: 大不列顛凱洛格有限公司 (“British Kellogg Company Ltd”) ko: 비스코스레이온 (“viscose-rayon”)

> jp: (*) 日本語 は 難しく ない zh: (*) 大不列顛 凱 洛 格 有限公司 ko: (*) 비스코스레 이온

by the ESP tokenizer. If `substring="2"` is configured for the current field, the following bigrams will be indexed:

> jp: (**) 日本 本語 語は は難 難し しく くな ない い__ zh: (**) 大不 不列 列顛 顛凱 凱洛 洛格 格有 有限 限公 公司 ko: (**) 비스 스코 코스 스레 레이 이온

This enables the user to find not only meaningful tokens identified by the tokenizer, but any even possibly meaningless substrings, such as (jp) “   ”, (zh) “   ”, (ko) “   ” which will likewise be split into the bigrams (jp) “ ” and “ ”, (zh) “ ” “ ”, and (ko) “ ” “ ”, thus matching the bigram version of the document.

A search for a unigram, e.g. (jp) し, (zh) 顛, (ko) 코 would be automatically converted into a wildcard search (jp) し*, (zh) 顛*, (ko) 코*, thereby matching the bigram (jp) しく, (zh) 顛凱, (ko) 코스. In other words, if N-gram search is configured, queries of length M < N will be transformed into wildcard queries allowing even the smallest substring of each token to be found.

Note: Note the pros and cons of Substring Search.

1. **Advantage:** It compensates for possible errors made by the automatic tokenizer.
2. **Disadvantage:** It leads to the insertion of many meaningless substrings to the index, the majority of which will never be searched for and are furthermore likely to cause a greal deal of "noise", i.e. irrelevant hits in the search results.
3. **Disadvantage:** It causes a significant increase in the size of the index.
4. And please see the section “Limitations of substring search” below.

Thus substring search should be used with caution and only if necessary. Note that if you wish to use substring search with composite fields, this must be specified separately in the definition of the composite field. Composite fields are defined independently of the fields they contain, hence a composite field with substring search enabled may contain fields for which substring search is not enabled. The reverse also applies.

Recommended values for N

If you would like to use substring search, an optimal value for N must be chosen. In **Japanese and Korean**, N=2 or N=3 generally yield good results; N=2 results in less index expansion, but slightly more complex queries. If the queries in the given application tend to be rather long, N=3 may be better, but if in doubt, benchmarking should be conducted using representative query logs or representative emulated queries before a final decision is made.

In Chinese, N=2 is generally better than N=3, as the average length of words in Chinese is 2.4 characters.

For more options regarding substring search (e.g. values N >= 33 if you do not want matches to be made across token boundaries), please consult the Configuration Guide.

Limitations of substring search

Substring search cannot be combined with certain other features. In particular, fields using substring search

- do not support dynamic teasers (highlighting of query terms in the result set)
- cannot be scope fields
- cannot be readily configured to use query side synonymy. Note that there are some workarounds. Please contact Solutions and Customer Services (SCS) regarding this.

Document processing pipeline

CJK specific features of the document processing pipeline.

Language and encoding identification

Recommended default / fall back configuration for the language detector stage.

In most web search applications, the language of each document is identified by the *LanguageAndEncodingDetector* stage. It provides for a parameter **FallbackLanguage** which should be set to either “zh”, “ja” or “ko” depending on the most frequent language in your collection.

The fallback language will apply whenever the language detector fails to identify the language of a document. This may occur, for example, if the document contains too little text.

A **fallback encoding** should also be defined. This is the encoding which is most commonly used in the documents to be indexed. For web search, this will usually be the following:

Principle language	Recommended fallback encoding
Simplified Chinese	GB18030
Standard traditional Chinese (limited support for Cantonese dialect characters)	BIG5
Traditional Chinese with the Cantonese extension as defined by the Hong Kong Government. BIG5_HKSCS is a superset of BIG5.	BIG5_HKSCS
Japanese	EUC-JP
Korean	EUC-KR

In applications other than web search, the language and encoding must be set correctly or identified for each document or database record.

Tokenizer and user dictionaries

Configuration options and user dictionaries for CJK specific tokenizers.

If the language field is correctly set, the appropriate language-specific tokenizer will automatically be applied to all fields configured with tokenize="auto" in the index profile.

The C++ interface provided within the tokenizer SDK should be used when adding an external tokenizer for a specific language. The plug-in must be registered by editing the main tokenization configuration file `$FASTSEARCH/etc/tokenizer/tokenization.xml`. For more information on how to do this, consult the documentation accompanying the SDK.

To improve the accuracy of the Chinese, Japanese and Korean tokenizers pre-installed in ESP, users can specify any character sequence which should always be treated as one token by adding the sequence to a language-specific *user dictionary*. The dictionaries are found in the following locations respectively.

szh	`$FASTSEARCH/etc/tokenizer/userdicts/zh-simp-user-dicts.utf8`
tzh	`$FASTSEARCH/etc/tokenizer/userdicts/zh-trad-user-dicts.utf8`
ja	`$FASTSEARCH/etc/tokenizer/userdicts/ja-user-dicts.utf8`

ko	`$FASTSEARCH/lib/basis/kma/dicts`

Chinese

Tokenizer configuration options for Chinese.

Chinese

There are two user dictionaries for Chinese: one for words written in simplified characters (zh-simp-user-dicts.utf8) and one for words written in traditional characters (zh-trad-user-dicts.utf8). This distinction is not important for the Chinese tokenizer, hence adding all new words to any one of these dictionaries will have the same effect as differentiating between the two sets of words. Nonetheless, separating the two sets of words may facilitate the administration of the user-defined words for some users.

The user dictionaries must be saved in the UTF-8 encoding standard. Each new word should be entered on a separate line and must not contain any space characters. These new tokens will take priority over any other words in the system dictionary. It should noted that the tokenizations specified in the user-defined tokens will be given preference regardless of whether such a tokenization is correct in a particular context.

```
矢向化
秋葉原
撲熱息痛
```

Japanese

Tokenizer configuration options for Japanese.

```
春夏秋冬
関西国際空港 2#4#6
```

The first line specifies that the string 春夏秋冬 shuold be treated as one single token, and hence never be split up into smaller units regardless of the context. The second line specifies that the string 関西国際空港 should be split into two tokens ending after the second, forth and sixth character respectively. This yields the tokens 関西, 国際, and 空港, independent of the context.

Note that there is currently no way of influencing the manner of **lemmatization** built into the Japanese tokenizer. The only way for users to custom-define lemmatization is to employ synonym expansion either on the document or query side in addition to the tokenization user dictionary. For example, the Japanese verb form ござらぬ should be lemmatized to ござる. This may not be supported by default, in which case the SBC, the search profile front-end, must be used to define a two-way synonym for the two terms. Note that the **language of the synonym dictionary** must be specified correctly while configuring the SBC, as this information will be used internally to tokenize the dictionary. If the language flag is set incorrectly, the dictionary entries will be tokenized incorrectly and synonym expansion will not behave as expected.

Korean

Tokenizer configuration options for Korean.

There are four configurable Korean user dictionaries in `$FASTSEARCH/lib/basis/kma/dicts`:

`ham-usr.dic`	New word dictionary: specifies previously unrecognized or incorrectly lemmatized words, e.g. 크리스토퍼, 소개팅, 가이드라인.
`ham-cnn.dic`	Compound noun dictionary: specifies the tokenization of compound nouns, e.g. 가용외환보유액 가용 외환 보유액.
`ham-asp.dic`	Spacing dictionary: specifies the tokenization of compounds other than noun compounds, e.g. 학교가자 학교 가자, 저푸른초원 저 푸른 초원.
`ham-rma.dic`	Lemmatization dictionary: specifies the lemma forms of words. This dictionary should only be used if adding the word to ham-usr.dic has no effect., e.g. 재건축을 재건축 + 을, 휴식처인 휴식처 + 이 + ㄴ .

The following requirements apply to all dictionaries:

- The .dic files must all be saved in EUC-KR (CP 949, KS C 5601) encoding
- The entries must be in Hangeul alphabetical order
- The dictionary may not contain more than 10,000 entries
- The total size of the user dictionary must not exceed 60,000 bytes
- Lines beginning with a semicolon (;) are treated as comments and are ignored

`ham-usr.dic` – New word dictionary

A word should be added to this dictionary if:

- a token in a compound is not correctly identified as it may not be in the system dictionary
- a word is not correctly lemmatized. This may be because its part of speech is incorrectly assigned by the Korean tokenizer

The new word with its part of speech should be entered in the following format:

```
[word][space][part of speech]
```

Part of speech codes:

N	noun 명사
U	dependent noun 의존명사
P	pronoun 대명사
c	compound noun 복합명사 (also add the word to compound noun dictionary `han-cnn.dic`)
T	transitive verb 타동사 (do not include the ending 다)
I	intransitive verb 자동사 (do not include the ending 다)
S	passive verb 피동동사 (do not include the ending 다)
C	causative verb 사동동사 (do not include the ending 다)
J	adjective 형용사 (do not include the ending 다)
B	adverb 부사

More than one part of speech can be assigned to one word by placing two or more POS codes next to each other, e.g. “IT”.

```
광주시 N
금목걸이 C
뒤얽히 T    (remember to omit "다" ending)
더해가 TI     (remember to omit "다" ending)
```

This adds the simple noun 광주시 (Gwangju City), the compound noun 금목걸이 (gold ring), the transitive verb 뒤얽히다 (to become entangled) and the transitive + intransitive verb 더해가다 to the list of known words.

`ham-cnn.dic` – Compound Noun Dictionary

Compound words which are not segmented correctly should be added to this file. Note that two letter words, e.g. 강산 cannot be further segmented into single letter words. Entries should be in the following format:

```
[word] [space] [length in characters of each token]
```

```
크린에어테크놀로지 45
사이버쇼핑센터 322
개량주의 0
```

This specifies that

1. 크린에어테크놀로지 (clear air technology) should be decomposed into 4 char + 5 char length tokens, i.e. 크린에어 (clear air) and 테크놀로지 (technology)
2. 사이버쇼핑센터 (cyber-shopping centre) should be split into 사이버 (cyber – 3 chars) 쇼핑 (shopping – 2 chars), 센터 (centre – 2 chars)
3. 개량주의 (reformism) should not be split up

Note:

- The entries must be in Hangeul alphabetical order
- A segmentation of "0" denotes that the word should not be split into smaller tokens
- Two letter words cannot be further segmented

`ham-asp.dic` – Spacing Dictionary

Korean words which are not compound nouns are sometimes written without a space between tokens. The segmentation of such phrases into their component words can be specified in this dictionary. Entries must be in the following format.

```
[word] [space] [2 x length in characters of each token]
```

```
노벨상의유래 8
저푸른초원 26
```

The examples specify that:

1. a space should be inserted after the forth character (8 / 2 = 4): 노벨상의 (Nobel prize’s) 유래 (origin).

2. a space should be inserted after the 1st character (2 / 2 = 1) and 3rd character (6 / 2 = 3): 저 (that) 푸른 (blue) 초원 (plain)

`ham-rma.dic` – Lemmatization Dictionary

If adding a word to the dictionary ham-usr.dic does not have the desired effect, the lemma form of a word can be specified explicitly for a particular inflectional form. Note that only the inflectional form entered will be lemmatized in theway specified. Other forms of the same stem will not be affected. Due to the algorithm used by the Korean lemmatizer, the rules specified in this file are not always given highest preference and hence do not have always have an effect. There are no alternatvie methods available for influencing the lemma form. Entries in this dictionary must be in the following format.

If the inflectional form does not have a particle 조사 or verb ending 어미:

```
[word] [tab] <stem, POS>
```

If the inflectional form has one or more particle 조사 or verb endings 어미:

```
[word] [tab] <stem, POS>, <ending, POS>
[word] [tab] <stem, POS>, <ending, POS>, <ending, POS> ......(as needed)
```

The POS's for the stems are the same as those used in ham-usr.dic. The POS of endings are:

e	final verb ending 어미
j	particle조사
c	verb "to be"이다

```
휴식처인 <휴식처, N> + <이, c> + <은, j>
재건축을 <재건축, N> + <을, j>
뭔지를 <무엇, P> + <이, c> + <ㄴ지를, e>
```

The examples specify that:

1. the lemma form of 휴식처인 is 휴식처 and ends with the verb "to be" 이다 plus the topic particle 은 ..
2. the lemma form of 재건축을 is 재건축 and ends with the particle 을.
3. the lemma form of 뭔지를 is 무엇 and ends with the verb "to be" 이다 plus the verb ending ㄴ지를.

Lemmatizer

CJK-related configuration options for the lemmatizer.

When the default CJK tokenizers are installed, the stage called "Lemmatizer" does not generate any lemma forms when CJK text is processed, as all forms of CJK lemmatization are carried out by the tokenizer stage. The lemmatizer stage simply rerouts the lemma forms added by the tokenizer to the appropriate fields of the document and query data. This lemma form rerouting is configured in the file `LemmatizationConfig.xml` in the stage `external_lemmatizer`. Removing this declaration will disable lemma retrieval for the language specified.

Query processing pipeline

This section describes CJK+-related options and configuration of the query processing pipeline.

Language Identification

When ESP processes queries, the language of each query must be specified correctly so that the appropriate tokenizer is selected for query processing.

There is no automatic language identification of queries. The language parameter must be set by the user or by the application sending the query for each query submitted. If it is not set, the default ("fallback") language will be used. This can be defined for each view in the SBC. If an application is primarily used to process CJK+ language queries, the fallback language should be specified based on which language is most commonly used to formulate queries.

In some instances, automatic language detection of queries is theoretically possible depending on the number and type of languages involved. This, however, may require help from Solution and Customer Services depending on the language combination.

Query Side Synonym Expansion

```


서강 대학교 → 서대, 서강대
```

Note: The tokens defined within a single synonym entry are always formulated as **PHRASE** queries, e.g.   PHRASE(,) where   and   must occur next to each other in that order in the query. Other query types such as **AND()** are not configurable.

Synonym expansion and substitution is not applied until *after* the query has been tokenized, hence the words in a user-defined synonym list must be tokenized in the same way as the query is. To ensure that this is carried out in the correct way, the **language flag of the synonym dictionary** must be set correctly.

Note that that only one language flag can be set for all the synonyms defined in a view in the SBC, hence all entries will be tokenized using the tokenizer of the selected language. It is possible to create a synonym list which contains a mixture of European languages words and **one** of the CJK+ languages. In this case, the language flag should be set to the CJK+ language. **Mixing more than one CJK+ language in the synonym dictionary of SBC is not recommended.** If you wish to mix CJK+ languages in the one synonym list, a custom-synonym stage stage should be defined for each of the languages in the qtf-config.xml.

Synonym Dictionaries Delivered in ESP

The following paragraphs provide a description of standard synonym dictionaries available for CJK query side synonym expansion.

Korean and Chinese Synonym Dictionaries

The Korean and Chinese synonym dictionaries mentioned hereunder are not pre-installed in ESP, but come with the Korean and Chinese Advanced Linguistic Services Pack. A FAST professional services consultant can provide you with this pack on request.

Korean translation dictionary

File name: `ko_foreign_word_translation.aut`

This synonym dictionary contains the Hangeul equivalent for English foreign words in Korean. Once installed, the Hangeul equivalent is added to a query when the English term is entered. Below are some examples.

English	Hangeul
air conditioner	,
Hawaiian shirts	알로하셔츠,알로하샤쓰
amplifier	앰플리파이어,앰프
centimeter	센티미터,센티
coordination	코디네이션,코디
undershirt	언더셔츠,언더샤쓰
white shirts	와이셔츠,와이샤쓰

Korean jeok-synonym dictionary

File name: `ko_jeok_suffixed_words.aut`

The Korean tokenizer does not remove the jeok suffix during lemmatization, hence the query " " (history) does not retrieve documents containing the word "   " (historical). If you would like search terms without the jeok suffix to also find documents with this suffix and vice versa, the jeok synonym dictionary should be installed. It automatically adds the jeok-suffixed word to the query and vice versa. Below are some examples of the entries in this dictionary.

Stem	Jeok-
가부장제	가부장제적
가설	가설적
가속	가속적
가시	가시적
가식	가식적
가정	가정적

Korean spelling variants dictionary

Spellchecking for Korean on page 124

Chinese Mandarin to Taiwanese Mandarin conversion dictionary

See section on Chinese character normalization(*Character Normalization for Chinese* on page 108).

Chinese misspellings to correct spellings conversion dictionary

Spellchecking Chinese on page 125

Installation of synonym dictionaries

1. Add the following instance to `$FASTSEARCH/etc/QRSERVER/webcluster/etc/qrserver/qtf_config.xml` .

```
<instance name="synonym_ko" type="external" resource="qt_synonym">
  <parameter-list name="qt.synonym" >
    <parameter name="enable" value="1"/>
    <parameter name="on_by_default" value="1"/>
    <parameter name="synonymdict1"
value=’"resources/dictionaries/synonyms/qt/ko_foreign_word_translation.aut"/>
    <parameter name="synonymdict2"
value=’"resources/dictionaries/synonyms/qt/ko_jeok_suffixed_words.aut"/>
  </parameter-list>
</instance>
```

2. Add the following lines to the `qt_pipeline` for which you wish to implement the synonym suggestions. The default `qt_pipeline`, “scopesearch” is used in the example below.

```
  <pipeline name="scopesearch" alias="esp5">
  <instance-ref name="utf8" critical="1"/>
  …
  <instance-ref name="simplefql"/>
  <instance-ref name="tokenize" critical="1"/>
  <instance-ref name="synonym_ko"/>
```

3. To activate the new configure, type the following at the command prompt `view-admin -m refresh -a`

Spellchecking (“Did you mean?”)

Problems and solutions for spellchecking in CJK languages.

General remarks (with examples from Japanese)

The main difficulty involved in spellchecking in CJK+ languages is that the character encodings used are very complex. Specifically, in UTF-8, the encoding system used by ESP, a single-character spelling mistake may involve 1, 2, 3, or more bytes of the query, depending on the encoding system and the specific character which is misspelled.

All queries passing through the query processing system of ESP are encoded in UTF-8. The spellchecking module (“QT DidYouMean”) automatically converts the query into the encoding used in the spellchecking dictionary, provided that the language of the query is correctly specified. As each language has its own spellcheck dictionary, a different character encoding may be used in each spellcheck dictionary.

The spellchecking dictionaries for European languages are each encoded in one of the ISO-8859-X encodings, hence each character corresponds to exactly one byte.

For Chinese, Japanese and Korean, the dictionary must be encoded in UTF-8. UCS-2, UCS-4, UTF-16 and UTF-32 cannot be used due to processing restrictions of the spellchecking QT module. In UTF-8, each Chinese, Japanese or Korean character generally corresponds to 3 bytes, although some characters may be comprised of 4 or more bytes.

Spellchecking is performed by looking up the query string token by token in the spellchecking dictionary. The lookup is implemented such that "fuzzy matches" can be retrieved, i.e. entries in the spellchecking dictionary which are slightly different from the original query are retrieved. In this way, spelling mistakes can be detected and alternatives can be suggested to the user.

If the standard configuration is used, the strings retrieved from the spellchecking dictionary may differ from the query token by up to 2 bytes. Therefore, in a UTF-8 CJK query, spelling mistakes that involve not more than 2 bytes of a token after tokenization can be detected. E.g. if in a Japanese Katakana string, one character is replaced by another, the correction can be made, because all Katakana characters happen to share the first 2 bytes in their UTF-8 representation, i.e. the difference will be 1 byte only. However, if a Kanji is replaced by another Kanji, the difference may (but need not) be 3 or more bytes, depending on the UTF-8 representation of the respective Kanji.

In other words, CJK spelling correction is generally possible if a UTF-8 encoded dictionary is used, but it will not behave in a consistent way (since the possible corrections depend on the UTF-8 representation of the strings involved).

This problem can be solved in several ways, each of which has certain advantages and drawbacks.

Solution I

Change the configuration such that mismatches of up to 3 bytes can be retrieved from the spellchecking dictionary. To do this, a copy of the language-specific configuration file `$FASTSEARCH/etc/config_data/QRServer/webcluster/etc/qrserver/didyoumean/spellcheck.english.xml` should be cated in the same directory. The copy can be given the name `spellcheck.japanese.xml` or something similar for other languages. Next, edit the file and set the`configuration.matcher.levenshtein.tolerance`parameter to 3 (instead of 2) and assign the name of your spellcheck dictionary `.ftaut` to the `configuration.matcher.levenshtein.automaton` Please refer to the Configuration Guide on how to create `.ftaut` files. Ensure the dictionary you create is encoded in UTF-8.

Next, edit the file `$FASTSEARCH/etc/config_data/QRServer/webcluster/etc/qrserver/qtf-config.xml`and add the necessary parameters to the instance definition of “didyoumean”. You will find a set of parameters for each language there; you may e.g. duplicate the parameters for “en” and change them to “ja”, “zh” or “ko” etc. The “matcher” parameter must be changed so that it points to the language specific configuration file created above. The “encoding” parameter should be changed to “utf-8”.

Following this, restart the QRServer(s), and confirm that spellchecking suggestions are generated for the respective CJK+ language.

The advantage of this approach is that spellchecking will function correctly for the vast majority of CJK characters, including most Hanzi (Kanji, Hanja), as most of them take 3 bytes of space. The drawback is that the spellchecking is not consistent (e.g. if two characters A and B share 1 or 2 bytes of their respective UTF-8 respresentation, while A and C share 0 bytes in a misspelled string XAY, the spellchecker will more likely suggest XBY as the correct string, rather than XCY, although XCY might be expected by the user).

Another potential problem with this solution is the that the performance of the QRServer (e.g. in terms of the number of queries processed per second) will decrease significantly, since fuzzy string matching at a tolerance level of 3 takes significantly more processing time than the default tolerance level of 2.

Solution 3

Use predefined correction rules instead of QT Didyoumean. Instead of building a dictionary of correctly spelled words, and applying fuzzy string matching to it, typical spelling mistakes may be captured in advance and added to a synonym dictionary. E.g. if the Japanese word “ ” is known to be a frequent misspelling of “ ” , one can add an entry to a synonym dictionary which maps “ ” to “ ”. This can be a one-way transformation or even a 2-way transformation, if it is not clear which of the two variants is actually correct - a common phenomenon in Japanese.

Entries can be added to the synonym dictionary using the SBC frontend, as documented in the Configuration Guide. Please note that it is absolutely necessary that users specify the language of the dictionary correctly, if CJK+ languages are involved. The dictionary should not contain words from more than one CJK+ language. However, words from several different European languages and one CJK+ can be safetly combined ; in this case, specify the CJK+ language as the language of the dictionary.)

Spellchecking for Korean

Two types of spellchecking are available for Korean: (i) the standard spellcheck "did you mean" module, (ii) a spelling variant synonym dictionary offering spelling suggestions. The two strategies are best combined to confer more extensive word coverage.

1. The "did you mean" spellcheck feature uses the standard spellchecking stage in ESP and is pre-installed in ESP 5.1. Similiar to spellchecking for European languages, Korean spellchecking suggestions are selected based on their frequency in general texts and on their degree of similarity to the misspelled word.
2. The second type of spellchecking offered is the normalization of variant spellings of foreign words ( ) into their standard form. This is implemented by way of a query expansion synonym dictionary. Below are a few examples:

Variant spelling	**Suggested spelling**
네트웍	네트워크
로봇	로보트
슈퍼맨	수퍼맨
인터네트	인터넷

Configuring Korean Spelling Variations

1. Add the following instance to `$FASTSEARCH/etc/QRSERVER/webcluster/etc/qrserver/qtf_config.xml`

```
<instance name="synonym_ko" type="external" resource="qt_synonym">
  <parameter-list name="qt.synonym" >
    <parameter name="enable" value="1"/>
    <parameter name="on_by_default" value="1"/>
    <parameter name="synonymdict1"
value=’"resources/dictionaries/synonyms/qt/ko_spelling_variation.aut"/>
  </parameter-list>
  </instance>
```

2. Add the following lines to the `qt_pipeline` for which you wish to implement the synonym suggestions. The default `qt_pipeline`, “scopesearch” is used in the example below.

```
  <pipeline name="scopesearch" alias="esp5">
  <instance-ref name="utf8" critical="1"/>
  …
  <instance-ref name="simplefql"/>
  <instance-ref name="tokenize" critical="1"/>
  <instance-ref name="synonym_ko"/>
```

3. To activate the configuration, type the following at the command prompt `view-admin -m refresh -a`

Spellchecking Chinese

There is no "did you mean" spellchecking module available for Chinese, as this module is not yet able to handle multi-byte characters to the extent needed to process Chinese Hanzi masterfully. Nevertheless, there is a spelling variant synonym dictionary which gives spelling suggestions when a user enters a misspelled word included in the dictionary. Two different dictionaries are available: one for spelling suggestions in Simplified Chinese characters and one for spelling suggestions in Traditional Characters. Below are some examples of spelling suggestions from the Traditional Chinese dictionary.

Misspelled word	**Suggested word**
仿傚	仿效
優遊	
傳盃換盞	傳杯換盞
傳布	
僥幸	僥倖
輩份	輩分
筆劃	筆畫

Configuring Chinese Spelling Varations

1. Add the following instance to `$FASTSEARCH/etc/QRSERVER/webcluster/etc/qrserver/qtf_config.xml`

```
<instance name="synonym_zh" type="external" resource="qt_synonym">
  <parameter-list name="qt.synonym" >
    <parameter name="enable" value="1"/>
    <parameter name="on_by_default" value="1"/>
    <parameter name="synonymdict1"
value='"resources/dictionaries/synonyms/qt/zh_spelling_variation_tzh.aut"/> for
Traditional Chinese
                   OR
    <parameter name="synonymdict1"
value='"resources/dictionaries/synonyms/qt/zh_spelling_variation_szh.aut"/>   for
Simplified Chinese
  </parameter-list>
  </instance>
```

2. Add the following lines to the `qt_pipeline` for which you wish to implement the synonym suggestions. The default `qt_pipeline`, “scopesearch” is used in the example below.

```
  <pipeline name="scopesearch" alias="esp5">
  <instance-ref name="utf8" critical="1"/>
  …
  <instance-ref name="simplefql"/>
  <instance-ref name="tokenize" critical="1"/>
  <instance-ref name="synonym_zh"/>
```

3. To activate the configuration, type the following at the command prompt `view-admin -m refresh -a`

Configure chinese spellcheck

1. Edit the file `$FASTSEARCH/etc/QRSERVER/webcluster/etc/qrserver/qtf_config.xml`
2. Add the following instance.

```
&lt;instance name="synonym_zh" type="external" resource="qt_synonym"&gt;
  &lt;parameter-list name="qt.synonym" &gt;
    &lt;parameter name="enable" value="1"/&gt;
    &lt;parameter name="on_by_default" value="1"/&gt;
    &lt;parameter name="synonymdict1" value='"resources/dictionaries/synonyms/qt/
zh_spelling_suggestions.aut"/&gt;
  &lt;/parameter-list&gt;
&lt;/instance&gt;
```

3. Add the following lines to the `qt_pipeline` for which you wish to implement the synonym suggestions. The default `qt_pipeline` , “scopesearch” is used in the example below.

```
&lt;pipeline name="scopesearch" alias="esp5"&gt;
&lt;instance-ref name="utf8" critical="1"/&gt;
…
&lt;instance-ref name="simplefql"/&gt;
&lt;instance-ref name="tokenize" critical="1"/&gt;
&lt;instance-ref name="synonym_zh"/&gt;
```

4. Save `qt_config.xml`
5. Type at the command prompt `view-admin -m refresh -a`

Fine tuning the CJK tokenizers

Configuration options for CJK language tokenization and lemmatization.

The tokenizers (morphological analyzers) for Chinese, Japanese and Korean which are built-into ESP, can be configured in several ways. All of the options are configured in the file `$FASTSEARCH/etc/tokenizer/tokenization.xml` For Chinese, Japanese, and Korean, this file provides for the definition of so called tokenizer plug-ins, i.e. dynamically linked external tokenizers. Depending on which plug-in is used, the options configurable will differ. ESP 5.1 is shipped with one plug-in for each of the three languages. The Chinese, Japanese and Korean tokenizers are called `basis-cma` , `basis-jma` and `basis-kma` , respectively.

Options for Japanese tokenization (basis-jma plugin)

This is the default specification of the `basis-jma` plug-in:

```
  <tokenizer-plug-in id="basis-jma" library="basis-jma-plugin"
  set-up="pooling"> <config> <basis-jma>
  <compound-analysis /> <dictionary-forms /> <lemmatization
  add-readings="true" /> </basis-jma> </config>
  </tokenizer-plug-in>
```

The XML elements included in the `basis-jma` main element refer to the various options made available by the standard plug-in. Should you decide to use your own plug-in for the tokenization of Japanese, the options listed below will no longer be available. To install your own custom tokenizer, you must use the C++ Tokenizer SDK. Custom-defined options would then have to be enabled through appropriate function calls in the code.

The following is a list of options available from the default plug-in. The “default” column refers to the settings of parameters if they are not explicitly mentioned in the configuration file.

Option	Description	Default	Example
`dictionary-path`	One or more locations of the dictionaries used by the Japanese tokenizer. The main dictionary is encrypted and cannot be modified, but the user dictionary can be used to define application-specific tokens.	$FASTSEARCH/lib/basis/ja/dicts/JP_%e.bin $FASTSEARCH/etc/tokenizer/userdicts/ja-user-dicts.utf8	`<dictionary-path>` etc/tokenizer/userdicts/ja-user-dicts.utf8 `</dictionary-path>`
separate-numbers-from-counters	If this is specified, combinations of digits and counters (助数詞) will be segmented as two separate tokens, e.g. 台 will be broken up into and 台 .	Specified.	<separate-numbers-from-counters>true</separate-numbers-from-counters>
separate-place-name-from-suffix	If this is specified, combinations of place names and “generic place type specifiers” will be treated as separate tokens, e.g. 神奈川県 will be split into 神奈川 and 県 .	Specified.	<separate-place-name-from-suffix>true</separate-place-name-from-suffix>
`dictionary-forms`	If this is specified as in the example, base forms of words will be added as “lemmatization variants” , i.e. the lemmatization feature of ESP will be activated.	Not specified (but it is explicitly mentioned in the default `tokenization.xml`)	`<dictionary-forms add-reading="true" />`
`compound-analysis`	This specifies the way components of compound words will be indexed. If this is enabled, e.g. 関西国際空港 will be split into 関西 , 国際 and 空港 .	The option has four attributes: separator: the character to insert between components. The default is a space character variant: name of the variant to be used in a scope search index to store the components. The default is 'C'. annotate-only: This is deprecated. create-lemmas: If set to “true” , base forms will be	`<compound-analysis separator=" " variant="C" create-lemmas="true" />`

Option	Description	Default	Example
		generated also for the components. This is “true” by default.	
`add-morphologic-info`	If this is specified, part of speech tagging will be enabled. This is necessary for certain advanced linguistic features, such as document vectorization optimized for Japanese.	Not specified.	`<add-morphologic-info>true</add-morphologic-info>`

Options for Korean tokenization (basis-kma plugin)

This is the default specification of the `basis-kma` plug-in:

Option	Description	Default
`dictionary-forms`	If specified as in the example, base forms of words will be added as lemmatization variants, i.e. the lemmatization feature of ESP is activated.	`<dictionary-forms/>`
`lemmatization`	2 possible attributes: `hanja-as-hangeul` : activates Hanja to Hangeul conversion `add-suffixes-as-tokens` : endings (조사,어미) are treated as separate tokens. This is necessary for dynamic hit highlighting. If deactivated, hit highlighting will not work.	`hanja-as-hangeul="yes" add-suffixes-tokens="yes"`
`compound-analysis`	This specifies the way components of compound words are indexed. If this is enabled, 국제민간항공기구 will, for example, be split into 국제 , 민간 , 항공 , 기구 . One attribute: `variant` : specifies the FIXML label for compounds in scope fields. The default is “C”	`variant="C"`

Option	Description	Default
`add-morphologic-info`	If this is specified, part of speech tagging will be enabled. These tags are not currently used in ESP 5.1.	`<add-morphologic-info>true</add-morphologic-info>`

```
&lt;tokenizer-plug-in id="basis-kma" library="basis-kma-plugin"
    set-up="pooling" &gt;
 &lt;config&gt;
  &lt;basis-kma&gt;
   &lt;compound-analysis variant="C"/&gt;
   &lt;dictionary-forms /&gt;
   &lt;suffix-splitting hanja-as-hangeul="true"
   add-suffixes-as-tokens="true"/&gt;
   &lt;add-morphologic-info/&gt;
  &lt;/basis-kma&gt;
 &lt;/config&gt;
&lt;/tokenizer-plug-in&gt;
```

Phonetic Search

This subchapter provides a description and configuration instructions for phonetic search for CJK languages.

Japanese

The standard `tokenization.xml` contains two versions of the Japanese tokenizer plug-in: `basis-jma` and `basis-jma-reading`. The latter uses a version of the external tokenizer which returns “readings”, i.e. Hiragana versions associated with each token. This feature can be used to enable phonetic search for Japanese. Note, however, that tokenization of purely phonetic queries, i.e. queries written solely in Hiragana, will frequently be inaccurate, as the tokenizer cannot handle words spelt in unconventional ways. Phonetic search for Japanese requires, therefore, a more careful analysis of the specific application for which it is to be used. This should be done in collaboration with FAST Solution and Customer Services. Thus, in this documentation, only the options of the standard `basis-jma` plug-in are covered (with no support of Hiragana readings).

Chinese

ESP 5.1 supports search queries written in Hanyu Pinyin and Bopomofo (Zhuyin). This means that users can enter the pinyin or bopomofo sequence of keywords as a search query and documents will be retrieved which contain character sequences whose pinyin matches the pinyin in the search query. For example, the search string “nuo4 ji1 ya4 shu4 ma3 xiang4 ji1” will return documents containing 諾基亞數碼相機.

Enable Pinyin Search

1. Edit the file `$FASTSEARCH/etc/tokenization/tokenizer.xml`
2. Search for the line `<!-- Declaration of the default mode -->`
3. You should see the following lines:

```
<!-- Declaration of the default mode -->

<default-tokenizer>
   <generic name="fds4-compat" />
   <normalization transformation="lowercase" canonical="true" />
</default-tokenizer>

<tokenizer language="zh zh-cn zh-simplified zh-tw zh-traditional" compatibility="true">

   <plug-in name="basis-cma" />
   <normalization transformation="lowercase" canonical="true" />
 </tokenizer>
```

4. Change the line `<plug-in name="basis-cma " />` in the text fragment above to: `<plug-in name="basis-cma-reading" />`
5. Next, search for the line `<!-- CMA with Pinyin readings -->` You should see the following lines.

```
    <!-- CMA with Pinyin readings -->
    <tokenizer-plug-in id="basis-cma-reading" library="basis-cma-plugin"
set-up="pooling">
      <config>
        <basis-cma>
          <reading type="HANYU_PINYIN_TONE_NUMBERS" variant="pinyin">
            <option>SEPARATE_SYLLABLES</option>
            <option>USE_V_FOR_U_DIAERESIS</option>
          </reading>
```

Options for Pinyin format

The element `<reading>` has three attributes:

`type`: This attribute can take the following values:

- `BT_HANYU_PINYIN_TONE_MARK`, e.g. “m i”
- `BT_HANYU_PINYIN_TONE_NUMBERS`, e.g. “mei3”
- `BT_HANYU_PINYIN_NO_TONES`, e.g. “mei”

`separator`: This defines which character is used between pinyin tokens. The default is the space character, i.e. `separator=" "`. It is best not to change this.

`variant` : The FIXML tag to use in the scopefields. The default is `pinyin`.

There are two sub-options for `<reading>`:

```
<option>SEPARATE_SYLLABLES</option>
```

The tokenizer will output the pinyin of documents with each pinyin syllable as a separate token, e.g. 安裝有線電視　“an1 zhuang1 you3 xian4 dian4 shi4”.

If this option is not set, the pinyin syllables of each word are written together, e.g. 安裝有線電視 “an1zhuang1 you3xian4dian4shi4”. This has the disadvantage that the user must know how the standard Chinese tokenizer tokenizes a text. In the example above, for example, most users would not expect 有線電視 to be tokenized as one word and would use the pinyin query “you3 xian4 dian4 shi4”. This would, however, not yield correct results. We suggest, therefore, that you set the option “separate syllables”.

```
<option>USE_V_FOR_U_DIAERESIS</option>
```

If this option is set, the pinyin of “ü” is “v”, e.g. 女 “nv”, not “nü”. This substitution is standard in most Chinese input methods, so many Chinese users exchange these letters automatically. This setting is therefore recommended.

To determine whether pinyin readings have been successfully activated, you should see something resembling the following line in the scope field

```
<catalog name="bscpxml">
```

of the FIXML of each Chinese document. The example below is taken from a document containing the text 美國旅行 Pinyin: “mei3 guo2 lv3 xing2”.

<context name="sentence"> <alts><alt>美國</alt><alt type="A">mei</alt></alts> <alts><alt>旅</alt><alt type="A">lv</alt><alt type="A">guo</alt></alts> <alts><alt>行</alt><alt type="A">xing</alt></alts>

Query format

The query must be written in a different format from standard simple queries, namely in FQL of the following form.

```
string("pinyin_syllable_1 pinyin_syllable_2", variant="pinyin", mode="onear", n= 3)
```

As this format is very cumbersome to enter, a qt_rewrite module can be installed which

1. automatically separates the syllables in a pinyin word, e.g. "zhong1guo2zhong3li3 wen1jia1bao3" is rewritten as "zhong1 guo2 zong3 li3 wen1 jia1 bao3".
2. converts the pinyin query into the correct FQL format

Once the qt_rewrite module is installed, all words in the query which should be interpreted as pinyin or bomopofo must be enclosed in

```
yin(....)
```

, for example:

```
    yin(wen1jia1bao3)
```

. The trigger "yin(...)" is configurable in qt_rewrite.

The qt_rewrite module is not pre-installed and must be downloaded from the Chinese Advanced Linguistics Service Pack. Please consult your FAST professional solutions consult for it.

Phonetic Search for Korean

Hanja 한자 to Hangeul 한글 conversion

Example: (Mao Zedong).

This feature enables users to search for Hanja words using their Hangeul equivalent. Hence, if a document contains the Hanja word 毛澤東, it is possible to search for this document by either entering 毛澤東 or 모택동 as the query search term. This feature is pre-installed in ESP and is activated by default.

Deactivate Hanja Conversion

1. Edit the `$FASTSEARCH/etc/tokenization/tokenization.xml`
2. Search for the string `hanja-as-hangeul="true"` and delete it. The remaining line should be: `<suffix-splitting add-suffixes-as-tokens="true"/>`
3. Save the file.
4. Restart the document processor(s).
5. Refeed all documents.

Known issues and workarounds

Various known problems of CJK processing, and recommended solutions / workarounds.

Tokenization of navigator fields

Problems related to the separator character of navigator fields being removed by the tokenizer.

In many applications, navigator fields are not extracted by ESP's entity extraction or vectorization modules, but are predefined in the content. Typically, such fields contain lists of terms, separated by a special boundary characters, such as ; or #.

This needs to be specified in the index profile, e.g. for a field myfield, as follows:

```
<field name="myfield" separator=";" index="no" tokenize="delimiters" />
```

In this case, an assumption was made that the field is only used to generate drill-down options for the user, but not to be searchable itself (`index="no"`). This is reasonable, if all the terms in the navigator are words which occur in the main document body (or whatever is the default index field), such that clicking one of the navigator entries will yield a non-empty result set, based on matches in the default index field.

The situation is different, if the navigator field contains terms which are not in general part of the main body field. In that case, the navigator field needs to be available for search, which means it has to be tokenized properly if it contains CJK+ words:

```
<field name="myfield" separator=";" index="yes" tokenize="auto" />
```

This, however, causes a problem: The CJK+ tokenizer will in fact consider the separator character ; as a token boundary ("SKIP character" according to the tokenizer configuration) and remove it. As a result, the navigator will not contain the expected drill-down options.

Preventing the tokenizer from removing the separator character of navigator fields

There is the following workaround for the problem described above.

Define a special mode in the tokenizer configuration which does not define ; (or whatever separator is used) as a SKIP character, but as a SPLIT character. Derive a tokenizer stage which applies the special mode to the navigator field in question.

1. In `$FASTSEARCH/etc/tokenizer/tokenization.xml`, create a mode that behaves almost like your default mode, but preserves the separator character.
   a) Make a copy of the file
   b) Identify the generic tokenizer used by the default tokenizer (of the default mode) and its character sets.
   c) Copy the `SKIP` character set of the original generic tokenizer and exclude the code-point of the separator character.

d) Copy the `SPLIT` character set of the original generic tokenizer or create a new one, if there is none and add the code-point of the separator character.
e) Copy the original generic tokenizer, but replace its `SKIP` and `SPLIT` character set by the two just created.
f) Create a new mode identical with the default mode, except that it uses the generic tokenizer created in as the default tokenizer.
g) Save the file and (in the case of a multi-node installation) distribute it to all nodes as `$FASTSEARCH/etc/tokenizer/tokenization.xml`.

2. Ensure that the document processors apply the new mode to the field in question.

a) Derive a processing stage from the existing “Tokenizer” document processor. The **Document Processing** section of the Admin GUI is best used for this.
b) In the **tokenize** parameter of the new processor you should be able to find a field specification `fieldname:fieldname` (where `fieldname` is the navigator field in question). Change this into `fieldname:fieldname:mode`, such that `mode` is the name of the tokenization mode.

3. Restart all nodes of the ESP installation, test the new set-up and refeed all content.

Entity extraction available for CJK languages

This section provides a description of the kinds of entity extraction available for CJK languages, and explains how to configure it.

Chinese

The Chinese entity extractors can extract the following types of entities.

Locations	Most countries and towns in China as well as the rest of the world
Companies	Famous Chinese and foreign companies
Dates	Western-style dates and traditional Chinese imperial dates
People’s names	(i) Chinese and foreigner’s names written in characters, (ii) Chinese Pinyin and dialect names written in the Latin alphabet. (ii) must be installed separately.

Note: There are two extractors for company names, locations and people’s names. One is for simplified characters, the other for traditional characters.

To install the entity extractors, select the **Document Processing** tab in the ESP Admin GUI, scroll down and edit the following stages:

`Personextractor0`

zh-simplified	`linguistics/extractors/configuration.personextractor.whitelist.szh.xml`
zh-traditional	`linguistics/extractors/configuration. personextractor.whitelist.tzh.xml`

`Personextractor1` Add the following strings to the field "matcher".

zh-simplified	`linguistics/extractors/configuration.personextractor.pass1.szh.xml`
zh-traditional	`linguistics/extractors/configuration.personextractor.pass1.tzh.xml`

Change the value of field `phrases` to “0”.

If people’s names are not identified by the location extractor, you can add them to the user-defined Chinese place name list (utf-8 encoding):

Traditional Chinese	`resources/dictionaries/matching/persons_acceptor.tzh.aut`
Simplified Chinese	`resources/dictionaries/matching/persons_acceptor.szh.aut`

If a word is marked as a person's name but should not be, you can add the word to the company name rejector list at:

Traditional Chinese	`resources/dictionaries/matching/persons_rejector.tzh.aut`
Simplified Chinese	resources/dictionaries/matching/persons_rejector.szh.aut

Company Extraction

The company extractor only recognizes large companies, such as the Fortune 500 companies and well-known companies listed on major stock markets. Other companies have to be added to the user-defined Chinese company name list (utf-8 encoding):

Traditional Chinese	`resources/dictionaries/matching/companies_acceptor.tzh.aut`
Simplified Chinese	`resources/dictionaries/matching/companies_acceptor.szh.aut`

Furthermore, a word is only recognized as a company name if it has some kind of suffix signalling that the word is a company, e.g. 有限公司 (Pty Ltd) 集團 (Corporation, Group) 銀行 (Bank). This is necessary as many company names are also found in product names, place names or even a person's name in some contexts.

`Companyextractor0`	**zh-simplified**	`linguistics/extractors/configuration.companyextractor.whitelist.szh.x`
	zh-traditional	`linguistics/extractors/configuration.companyextractor.whitelist.tzh.x`
Companyextractor1	Add the following string to the field "matcher"	
	zh-simplified	`linguistics/extractors/configuration.companyextractor.pass1.szh.xm`
	zh-traditional	`linguistics/extractors/configuration.companyextractor.pass1.tzh.xm`

If a word is marked as a company name but should not be, you can add the word to the company name rejector list at:

Traditional Chinese	`resources/dictionaries/matching/companies_rejector.tzh.aut`
Simplified Chinese	`resources/dictionaries/matching/companies_rejector.szh.aut`

Location Extraction

Most locations are only tagged as locations if there is some suffix indicating that they are used as place names in that particular context, e.g. with 山, 市, 省, 洲. Many location words also occur in contexts where they denote the culture, cuisine or style of something or are simply part of a name, e.g. Chinese restaurant, Bank of England, Russian Salad.

`Locationextractor0`	**zh-simplified**	`linguistics/extractors/configuration.locationextractor.whitelist.szh.`
	zh-traditional	`linguistics/extractors/configuration.locationextractor.whitelist.tzh.`
`Locationextractor1`	Add the following string to the field "matcher".	
	zh-simplified	`linguistics/extractors/configuration.locationextractor.pass1.szh.x`
	zh-traditional	`linguistics/extractors/configuration.locationextractor.pass1.tzh.x`

If place names are not identified by the location extractor, you can add them to the user-defined Chinese place name list (utf-8 encoding):

Traditional Chinese	`resources/dictionaries/matching/locations_acceptor.tzh.aut`
Simplified Chinese	`resources/dictionaries/matching/locations_acceptor.szh.aut`

If a word is marked as a place name but should not be, you can add the word to the company name rejector list at:

Traditional Chinese	`resources/dictionaries/matching/locations_rejector.tzh.aut`
Simplified Chinese	`resources/dictionaries/matching/locations_rejector.szh.aut`

Date Extractor

Add the following string to the field "matcher".

```
zh:linguistics/extractors/configuration.dateextractor.zh.xml
```

Korean

The Korean entity extractors can extract the following types of entities.

Locations	Countries and towns in Korea as well as the rest of the world
Companies	Korean and foreign major companies
Dates	Western-style dates and traditional Korean imperial dates

To install the extractors, follow the steps outlined for European languages.

Japanese

People’s names

This includes Japanese, Chinese, and Korean person names written in Kanji, as well as foreign names written in Katakana. It does not include names written in Romaji. This can be enabled as part of a project, together with FAST Solution and Customer Services. The person name extraction is based on a statistical algorithm as well as context-sensitive rules. It is not directly based on a dictionary of names, thus it is not possible to readily add names to that dictionary. However, if a certain list of names should be extracted, it is possible to define a separate extraction stage, configured as a “Verbatim Matcher”, using the predefined list as a dictionary.

Place names

This includes country and city names in Japan or elsewhere in the world, writing in Kanji or Katakana. This uses a dictionary internally, which however is encrypted and cannot readily be read of modified by technicians other than members of FAST SCS. If a customer-defined dictionary should be used, it is recommended to set up a separate extraction stage, configured as a “Verbatim Matcher”.

Dates

Dates written in common Japanese formats (such as 平成元年 月 日) are recognized and transformed into the standard ISO date format. This includes the processing of imperial years back to the 11th century AD. The patterns and normalization algorithms used for this are encrypted and binary, they cannot readily be changed. Should this be necessary, please ask for the help of FAST Technical Support. The only exception to this is the normalization of imperial years (元号). The file `$FASTSEARCH/etc/resources/matching/japanese/gengoumap.txt` contains all era names and the year (元年) when each began. It starts with 天養元年 (1144) and covers all later era names completely. If era names before 1144 are to be added, the file can simply be edited (note that it needs to be saved in UTF-8, with Linux-style line endings). After modifying it, it needs to be copied to all processor nodes (in a multi-node installation), and all document processors need to be restarted.

Prices

Prices in common currencies, including Yen, are recognized. The currency symbols are normalized, esp. 円 is transformed to . Numbers written in Kanji are normalized as well, e.g. 億 万 千 百二十五円 will be transformed to 200203426 . The patterns and normalization algorithms used for this are encrypted and binary, they cannot readily be changed. Should this be necessary, please ask for the help of FAST Technical Support.

Document Vectorization, Find-Similar and unsupervised clustering in Japanese

This section provides a description of configuration options available for Japanese document vectors. It discusses how different options apply to different purposes, such as find-similar or unsupervised clustering.

Many of the standard document processing pipelines (esp. the `Generic Pipeline` and the `Semantic Pipeline` contain a stage `Vectorizer` which extracts a vector of terms and phrases from the document which are supposed to be very specific and very descriptive of its content. These terms are usually noun phrases or subject-specific terminology. For each term, a weight is computed, based on the frequency of the term in the document and in other documents (TF-IDF), or using other methods, depending on configuration parameters. There is general documentation of this in the Configuration Guide, and all of it applies to CJK+ languages as well.

However, for the method used to identify the terms that should be extracted, there are special options available for Japanese.

Term Selection

The configuration file for the Japanese vectorizer is located at `$FASTSEARCH/etc/config_data/DocumentProcessor/linguistics/vectorizer/configuration.matcher_medium_precision.ja.xml`

But there are two alternative configuration files which can be used instead of the default one – both of them are located in the same directory, their names are `configuration.matcher_high_precision.ja.xml` and `configuration.matcher_low_precision.ja.xml`.

In other words, there are three modes available for Japanese vectorization:

High precision	This extracts only terminology that ends with a certain typical word, such as 学 性 症 程度, and if the words preceeding the typical words are nouns or adjectives following certain syntactic rules typical of Japanese. E.g. words such as 光学 虚血性 不眠症 値段程度 will be extracted.
Medium precision	This also extracts only terms ending in certain known, typical patterns, but it is more flexible regarding the characters / words preceeding the typical ending. E.g. it allows for any sequence of Kanji (not only noun phrases) to be at the beginning of the extracted term.
Low precision	This extracts any sequence of adjectives and nouns that ends in a noun.

Note:

All the term extraction makes use of part of speech tagging (品詞タッガー), which is performed as part of tokenization for Japanese documents if the `add-morphologic-info` parameter is set in the tokenizer configuration (which is the case by default). See also the section about configuring the tokenizer.

The part of speech tagger is used as a black box by ESP and can not be configured or modified in any way. However, an external tagger can be integrated using the tokenization SDK (and associated C++ based API).

Changing the vectorization mode

Of the three vectorization modes (high, medium, low precision), the medium one is used by default. This can be changed by editing the `matcher` parameter of the `Vectorizer` stage (using the **Document Processing** section of the Admin GUI). This parameter contains the filename of the matcher to be used, it can be changed to point to the configuration file for the high or low precision vectorizer.

The filename to be used are as follows:

Low	`linguistics/vectorizer/configuration.matcher_low_precision.ja.xml`
Medium (default)	`linguistics/vectorizer/configuration.matcher_medium_precision.ja.xml`
High	`linguistics/vectorizer/configuration.matcher_high_precision.ja.xml`

After changing this, all content needs to be re-fed in order for the change to take effect.

Which mode to use

Which mode is best used for the vectorizer depends on what the vectors are used for.

If the main purpose is to provide a navigator for drill-down, either the high or medium precision mode should be used. The high precision mode will provide fewer, but more specific terms, while the medium mode will provide more, but partially less specific terms.

If the main purpose is "find similar" or unsupervised clustering, the low precision mode should be used.

Find similar

In the results returned for a query, each item X is associated with three hyperlinks related to the find similar function: **Refine similar**, **Find similar**, and **Exclude similar**. The first one limits the search result to those documents that are similar to X, the second one issues a new query for all documents similar to X (not only those that are part of the current result list), the third one limits the current result list to all those that are **not** similar to X.

The notion of similar documents is defined as documents that have a similar vector, i.e. that share some of the specific terminology extracted for the document vector, with similar weights. Hence what is retrieved as similar documents depends strongly on the configuration of the vectorizer. For Japanese, it is recommended to use the low precision mode of the vectorizer, please see the previous section on how to configure this.

Unsupervised clustering

Another standard feature of ESP 5.1 is unsupervised clustering, i.e. the automatic grouping of the documents of the result set into clusters of semantically related documents. This is based on a notion of document similarity, just like described in the previous section. For this, it is also recommended to use the low precision mode of the vectorizer.

Query by example

Instead of finding documents similar to one of the documents in the result set, ESP 5.1 also enables the user to type (or copy and paste) a document or any sort of text directly into the search frontend, and to retrieve documents similar to this. This is performed by dynamically applying vectorization to the text entered by the user, thus to extract a document vector from that text and to use it.

Important: This feature is not supported for Japanese, Chinese and Korean documents.

Query Processing

Particular issues with CJK query processing.

Implicit Phrasing

CJK+ queries are automatically tokenized by the language-specific tokenizer. E.g. the query (ja) 東京の天気, (zh) 東京的天氣, (ko) 국회상임 위원회 will be segmented into three tokens (ja) 東京、の、天気, (zh) 東京 的 天氣, (ko) 국회 상임 위원회. By default, the result of this is regarded as a phrase query, i.e. the system will behave as if the user had typed the query

```
(ja) 東京 の 天気
(zh) 東京 的 天氣
(ko) 국회 상임 위원회
```

(note the double quotation marks). These queries will only return documents that contain the exact same sequence of tokens as in the query. Documents which contain the three words with other tokens intervening will not be retrieved.

This behaviour may be a problem for some applications, especially if many queries are expected to contain non-whitespace separated lists of terms, e.g. (jp) “ ”, (zh) “ ”. The user who enters such queries most likely wishes to find documents containing all the search terms in a document, however not necessarily in that exact sequence. Such queries are best transformed into AND queries:

```
(jp) AND(東京都,千葉県,埼玉県,神奈川県)
(zh) AND(上海市,靜安區,閔行區,虹口區)
```

This transformation can be performed automatically by modifying the QRServer configuration. The file `$FASTSEARCH/etc/config_data/QRServer/webcluster/etc/qrserver/qtf-config.xml` must be modified, specifically the stringmode parameter must be changed to "AND":

```
<parameter name="stringmode" value="AND"/>
```

(The default value is `PHRASE`, indicating that CJK+ tokenized queries should be considered as phrase queries as described above.)

Wildcard queries and CJK+

If, for a certain field, the wildcard parameter is configured as

```
<field name="myname" wildcard="full" />
```

in the index profile, queries containing wildcard operators `*` and `?` anywhere in the query will be supported. For example, a query of the form

```
A*B
```

will retrieve results containing **tokens** that contain A as a prefix and B as a suffix. However, this is strictly limited to tokens and so does not sentences or phrases, starting with A and ending in B.

This may cause some confusion when dealing with CJK+ languages, as which documents are actually returned for a given wildcard query is highly dependent on the way these documents are tokenized. For example, two Japanese documents

```
ja1: 美術館
ja2: 美しい博物館

zh1 美術館
zh2 美國的博物館

ko1: 미술관
ko2: 미국박물관
```

which are tokenized as

```
ja1: 美術館
ja2: 美しい 博物館

zh1 美術館
zh2 美國 的 美物館

ko1: 미술관
ko2: 미국 박물관
```

respectively, behave differently when the query

```
(ja,zh) 美 館
(ko) 미*관
```

is submitted. ja1/zh1/ko1 will match, but ja2,/zh2/ko2 will not, because in ja2/zh2/ko2, the (ja,zh) 美, (ko) 미 is not part of the same token as (ja,zh) 館, (ko) 박물관.

Hence, when using wildcard search with CJK+ languages, it should be noted that the `?` and `*` operators denote wildcard characters within the one token, not characters spanning tokens.

Chapter 14

Dictionaries

Topics:

Linguistics modules in ESP rely on data stored in comprehensive task specific dictionaries.

Most dictionaries types in ESP can be edited to customize the search experience. For example, you can create and edit synonym dictionaries to provide expansions suited to your content. You can also add and modify lemmatization and spellcheck dictionaries to cover words that are important in your search context. Furthermore, you can customize entity extraction to suit the needs of of your application. The following tools are available to edit dictionaries and configure linguistic processing in ESP:

- `LinguisticsStudio` is a powerful GUI tool which can be used to create and edit most dictionary types in ESP and deploy them to the system. You can also use it to configure lemmatization and other linguist features. `LinguisticsStudio` offers extensive online help.
- `Dictman` is a command line interface which can be used to edit most dictionary types. `Dictman` commands are described in *Dictman Command Reference* on page 144 in this guide.
- `dictcompile` offers the functionality to compile text and xml files into the ESP internal dictionary formats.
- The `Search Business Center (SBC)` can be used to set up search profile specific synonyms as well as boost and block lists for query terms. Refer to the *Fast Home and Search Business Center Guide* to learn more about this tool.

The choice of tools depends on the user requirements. Generally, `LinguisticsStudio` is the best choice if you want to manually edit existing dictionaries or create new ones. `Dictman` or `dictcompile` are best suited to batch processing.

Scenarios for editing and compiling dictionaries

Make use of the dictionary editing and compilation tools for the following usecases:

- You need to modify existing dictionaries to suit your data or to set up special search requirements
- You want to add your custom dictionaries for an additional language or piece of functionality
- You have changed the tokenization setting and have to recompile synonym and lemmatization dictionaries (required)

Editable Dictionary Types

Commonly used dictionary formats with the names and abbreviations for `dictman` and `dictcompile` include:

Type	Abbreviation	Full name	Description
Simple spell checking	SPW	SPELLCHECK_WORDS	Language-specific spell checking dictionaries, containing single words.
Advanced phrase spell checking	SPP	SPELLCHECK_PHRASES	Spell checking on phrases (advanced spell checking). Generally not language specific.
Spell checking exceptions	EXC	SPELLCHECK_EXCEPTIONS	Exceptions for spell checking (advanced and simple spell checking)
Antiphrases	AP	ANTI_PHRASES	
Query side synonyms	SDQ	QUERY_SYNONYM_EXPANSION	Synonym dictionaries for queries
Document side synonyms	SDD	DOCPROC_SYNONYM_EXPANSION	Synonym expansion dictionaries for documents
Lemmatization	LD	LEMMATIZATION	Language-specific lemmatization dictionaries
Lemmatization blacklist	LB	LEMMATIZATION_BLACKLIST	Language-specific blacklists for lemmatization
Offensive content	OC	OFFENSIVE_CONTENT_DETECTION	Offensive content detection (containing all languages for OCF in one dictionary)
Entity extraction	ED	ENTITY_EXTRACTION	Entity extraction dictionaries (both blacklists and whitelists)
Vectorizer	VE	DOCUMENT_VECTORIZER	Vectorization dictionaries
Generic	GD	GENERIC	All-purpose format for custom dictionaries

The remaining, less commonly used dictionary formats are:

Type	Abbreviation	Full name	Description
Index frequency	FL	INDEX_FREQUENCY	Used for spell checking optimization.
Lemmatization (reduction)	LDR	LEMMATIZATION_REDUCTION	Rarely used. Commonly a general lemmatization dictionary (see above) is set up, which can be compiled into a reduction and expansion dictionary.
Lemmatization (expansion)	LDE	LEMMATIZATION_EXPANSION	Rarely used. Commonly a general lemmatization dictionary (see above) is set up, which can be compiled into a reduction and expansion dictionary.
Linguistic master	LM	LINGUISTIC_MASTER	Development purposes only.

In `dictman`, you can display all available dictionary type abbreviations by typing `types` at the command line.

In `dictman`, when listing all dictionaries with `ls`, dictionary types are displayed using the abbreviations. You can also use the abbreviated types for all commands that require an indication of the type. You can display all dictionaries of one type by typing `ls TYPEABBREVIATION`

Dictionary Maintenance with LinguisticsStudio

`LinguisticsStudio` is a powerful GUI to edit ESP dictionaries for all linguistics modules.

`LinguisticsStudio` is a Java-based application running on the Eclipse Rich Client Platform. It can run on a different machine than ESP and connects to an ESP installation via the network.

`LinguisticsStudio` allows one to edit most linguistics dictionaries using a graphical user interface. It includes tabular editors for all dictionary types and deployment tools. `LinguisticsStudio` supports the same dictionary types as Dictman, and allows one to import the same dictionary formats as required by the `import` command of Dictman and for Dictcompile. It also supports importing dictionary sources from Excel- and CSV-files.

Dictionaries edited with `LinguisticsStudio` can be automatically deployed to your ESP installation.

Refer to the `LinguisticsStudio` application's help functions to use this tool.

To learn more about dictionary types and formats, please refer to the overview of dictionary types and the reference for dictionary editing and import in the `dictman` chapter *Dictionary Maintenance with Dictman* on page 144. All types referenced in this chapter can also be edited with `LinguisticsStudio`.

Dictionary Compilation with Dictcompile

Dictcompile is a tool that allows compilation of dictionaries of various types to be used in ESP from text sources in a predefined format.

Dictcompile uses the same dictionary types as Dictman, and uses the same dictionary formats as required by the `import` command of Dictman.

Dictionaries compiled with dictcompile are not automatically deployed to the right locations and have to be copied manually.

If you want to edit the dictionaries using a graphical user interface and you need deployment and configuration support for your dictionaries, consider using `LinguisticsStudio`.

Compiling a Dictionary

Refer to *Modifying Dictionaries with dictman* on page 148 to learn more about the input format for dictionaries. The dictionary import format for `dictman` is the same as that for `dictcompile`.

To compile with dictcompile, you will need to create text source files in the dictman/dictcompile import format. Dictionaries are compiled into `.aut` or `.ftaut` (spell checking only) files.

The compiled dictionaries have to be deployed manually to all of the relevant folders on the affected nodes.

- all search nodes in case of query side dictionaries
- all document processing nodes in case of document side dictionaries
- both in case of query and document side dictionaries (e.g. spell checking)

1. Copy the source you want to compile to the machine where you want to work with dictcompile.
2. Go to `$FASTSEARCH/bin` and set the environment variables.
3. Compile the dictionary by running dictcompile:

```
./dictcompile.sh -i INPUTFILE -t TYPE -A ENCODING -d DELIMITER -o OUTPUTFILE -R
PROPERTIES --verbose
```

Compiling a spell checking dictionary from the source file myspellcheck_utf-8.4ftaut, encoded in the UTF-8 encoding with the output encoded in ISO-8859-1 (the usual encoding for spell checking for Western languages:

```
./dictcompile.sh -i myspellcheck_utf-8.4ftaut -t SP -A ISO-8859-1 -d "\." -o
myspellcheck_iso-8859-1.4ftaut --verbose
```

Compiling an expansion and reduction lemmatization dictionary from source mylemmas.4aut, encoded in UTF-8 Note that two compiled dictionaries have to be specified here - one for expansion, the other for reduction.

```
./dictcompile.sh -i mylemmas.4aut -t LD -d "\." -o mylemmas_exp.aut mylemmas_red.aut
--verbose
```

Dictionary Maintenance with Dictman

The esp4jtool-dictman dictionary manager is a command line shell that allows you to create, browse and edit the dictionary types that are used by different linguistic components within the FAST ESP processing and query pipelines.

Dictionaries are editable data structures of name-value pairs used for such features as phrasing, anti-phrasing, lemmatization, spell checking, synonym expansion, offensive content filtering, and vectorization.

Through esp4jtool-dictman, dictionaries can be edited and later compiled into compiled dictionaries also known as automata, which are non-writable dictionaries optimized for fast read access. These compiled dictionaries serve as input for stages in the query and document processing.

The same dictionary formats are used by `dictcompile`

Dictionaries are always held as master copies on the node on which the DictionaryService runs, and may be compiled (and later deployed) from this place. esp4jtool-dictman provides a command shell similar to a simple ftp or database client.

Dictman Command Reference

The esp4jtool-dictman tool can either run interactively or as a batch processor. If no filename is given, esp4jtool-dictman will run in interactive mode; that is, a prompt is displayed. If a filename is sent as a parameter, esp4jtool-dictman will execute the commands in this file and exit. Responses will be printed to stdout.

The esp4jtool-dictman tool runs in one of the following operating modes:

- as a batch processor executing the commands given in the file specified with the -f option.
- as a batch processor executing the commands given on command line.

  ```
  Example: dictman -e "info AP indep_antiphrases"
  ```
- as a command line shell if -f and -e are not specified.

In each case, esp4jtool-dictman will try to connect to an adminserver instance defined in the `$FASTSEARCH/etc/esp4j.properties` (esp.adminserver.host, esp.adminserver.port).

If a different host is given through the --esphost option, then esp4jtool-dictman will try to connect to the specified host, but will still try to read the port number from this file unless --espport is given.

Starting dictman

1. Set environment variables.

```
On Linux:
cd $FASTSEARCH/bin
>source ./setupenv.sh
```

```
On Windows:
<your FAST ESP installation directory>\bin\setupenv.bat
```

2. Start the dictman console:

```
On Linux:
> dictman.sh
On Windows:
> dictman
```

Command Line Option

Usage:

```
esp4jtool-dictman: options
```

The following table lists the esp4jtool-dictman command line options:

Table **2**: esp4jtool-dictman Command Line Options

Option	Description
-d,--properties <properties>	Application properties, syntax: -d <key1>=<value1> <key2>=<value2> ... .
-B,--espbaseport <baseport>	Baseport for FAST ESP installation.
-D,--debug	Debug mode. DEBUG level logging to CONSOLE appender.
-E,--help	Print this help message.
-H,--esphost <hostname>	Host for FAST ESP adminserver.
-L,--verbose	Verbose mode. INFO level logging to CONSOLE appender.
-O,--espport <port>	Port for FAST ESP adminserver. The port is the base port as issued to the installer.
-P,--pass <password>	Password for FAST ESP adminserver.
-U,--user <user name>	Username for FAST ESP adminserver.
-V,--version	Print version.
-e,--execute <execute>	Execute commands and exit.
-f,--file <file>	Execute commands in batch file and exit.
-l,--local	Use esp4jtool-dictman locally with name of the resource directory as specified.

How to Display Help in Dictman

1. Start dictman
2. To display a list of available commands, type `?` and press the enter key.
3. To display help for a particular command, type `COMMAND ?` and press the enter key.
   `command (? for help)> compile ?`

Entering Commands

When entering a command, its arguments are tokenized according to the following rules:

Usually a single word is one token: The string:

```
command (? for help)> insert Los Angeles
Inserting into [ENTITY_EXTRACTION(ED),locations]
inserting 2 entries
```

will insert two keys, "Los" and "Angeles" in the dictionary.

```
command (? for help)> queryquerying for '', returning max. 50 entries
starting from offset 0
Angeles X
Los     X
returned 2 entries
```

if a token contains spaces, it can be grouped using double quotes (").

```
command (? for help)> insert "Los Angeles"
Inserting into [ENTITY_EXTRACTION(ED),locations]
inserting 1 entry
```

will insert one key named "Los Angeles" in the dictionary.

```
command (? for help)> query
querying for '', returning max. 50 entries starting from offset 0
Angeles     X
Los         X
Los Angeles X
returned 3 entries
```

An "=" also works as a divider, separating keys from values:

```
command (? for help)> insert "n y"="new york"
Inserting into [ENTITY_EXTRACTION(ED),locations]
inserting 1 entry

command (? for help)> query
querying for '', returning max. 50 entries starting from offset 0
Angeles     X
Los         X
Los Angeles X
n y         X new york
returned 4 entries
```

No spaces should be placed between the single structural parts of a complex value. In case of a QUERY_SYNONYM_EXPANSION dictionary do not enter:

```
command (? for help)> insert calculation=[[false],[ [[
arithmetic,1,false,true]]]]
```

This is a correct command.

```
command (? for help)> insert
calculation=[[false],[[[arithmetic,1,false,true]]]]
Inserting into [QUERY_SYNONYM_EXPANSION(SDQ),synonyms]
inserting 1 entry
```

- Commands end at a new line or a ";" character.
- Commands can be be spread across multiple lines by entering a "\" character at the end of a line. The first line without a backslash denotes the end of the command:

  ```
  command (? for help)> insert one \
  > two \
  > three
  command (? for help)>
  ```

Executing Shell Commands

You can execute a shell command by preceding it with a "!" character:

```
command (? for help)> !pwd
executing 'pwd'/home/fastsearch/datasearch
finished
command (? for help)>
```

Value Formats

Some dictionaries contain structural information encoded in their values. In order to facilitate entering these values, esp4jtool-dictman uses a coherent syntax for specifying these values where:

```
Value ::= String | List
List ::= "[" ListEntriesOrEmpty "]"
ListEntriesOrEmpty ::= ListEntries | <E>
ListEntries ::= Value | Value "," ListEntries
String ::= an arbitrary string with "[", "]" and "," escaped by "\"
```

LemmatizationDictionaries, for example, contain lists of strings:

`[string1,string2,string3]`

SynonymDictionaries map keys to a more complex structure.

Make sure there are no spaces between the single structural parts of a complex value.

Leaving the Command Shell

Use the quit command to exit the command shell. This command will save, close and unlock all opened dictionaries by the current user on the server, and exit.

Tip: It is not recommended to press Ctrl-C in order to exit esp4jtool-dictman. The dictionaries will not be closed properly and may be time consuming the next time they are opened.

Batch Processing

Dictman can be used for batch processing where dictman commands are written in a script and submitted as arguments to the dictman launcher.

You can execute commands by issuing them to the esp4jtool-dictman command in the same way that you do for the command shell.

Commands can be executed using the -f switch:

```
$ sh dictman -U foo -P bar -f commands.txt
```

In this case they will be interpreted line after line, in the same way as they would in the command shell.

- When using batch processing, the end of the batch terminates the client.
- A batch will be stopped if an error condition occurs.
- The command syntax is the same as if you entered them via the shell. As in many scripting languages, you can use a # sign to start a line comment.

Example

To import a text file containing lists of spelling variations into a new document processor side synonym dictionary, with keys and values separated by a "." character, a sample line would be:

```
example1.[example 1,ex1,ex 1]
```

The command file would be:

```
{
  sdd spellvars      # alias for "with sdd spellvars"
  create en         # create this dictionary (type: synonym dict. DP side
 , language: English)
  import ./spellvars.txt "\." ISO-8859-1   # "." Delimiter has to be
escaped
  optimize           # optimize master dictionary for read access
  clb                # generate the automaton
}                    # alias for "endwith"
```

Modifying Dictionaries with dictman

The `dictman` dictionary manager works with dictionaries that are located in the resource repository; it does not work directly with the $FASTSEARCH/resources/dictionaries/.

The esp4jtool-dictman tool replaces the dictctrl tool used in previous versions of FAST ESP.

Tip: Refer to the Dictman reference for usage information including syntax and options information for the esp4jtool-dictman tool.

In general, if you want to modify the dictionaries in the resource repository with esp4jtool-dictman:

- start esp4jtool-dictman
- edit the dictionaries in the source format

- • compile the modified dictionaries into binary form
  - copy them to the FASTSEARCH/resources/dictionaries/ location

  If you have modified an index side dictionary:

  - Restart any Document Processing Servers to deploy the new modified dictionaries
  - Re-feed the documents you want to be affected by the change.

  If you have modified a query side dictionary:

  - Restart the qrserver to deploy the new modified dictionaries

Properties for Dictionaries

Editable dictionaries can have properties that affect their handling and compilation.

In dictman, dictionary properties can be set with the `setproperty` command, and can be looked up with the `info` command.

In dictcompile, dictionary properties can be set with the `-R` switch.

The following table lists dictionary properties that can be set by the user, and describes their effects.

Property	**Values**	**Relevant for dictionary types**	**Description**
doTokenize	true,false	all	If set to `true`, the dictionary is tokenized before compilation, using the tokenizer configuration set for the system. This flag should be set to `true` for lemmatization, synonym and OCF dictionaries, which work on tokenized content. It should be set to `false` for entity dictionaries.
keyEncoding,valueEncoding	IANA encoding specification (e.g. ISO-8859-1,UTF-8)	all	The encoding of the compiled dictionary (keys and values). UTF-8, except for spell checking dictionaries, which are always encoded in a single byte encoding. Set to ISO-8859-1 for the values of lemmatization dictionaries for format purposes (latter should not be changed).
language	ISO-639 language code	all	Sets the language of the dictionary. This affect the tokenization, in case the dictionary is tokenized before compilation.
generateExpansion, generateReduction	true,false	lemmatization dictionaries only	Determines the type of compiled dictionaries which are generated. Generally, both expansion and reduction dictionaries are required, and both flags should be set to true.
description	any string	all dictionaries	Contains an optional description of the dictionary.

Editing, Importing and Compiling Dictionaries

The following commands can be used to edit dictionaries of all types. The examples below are for a spellcheck dictionary.

1. Open dictman.
   ```
   dictman.sh
   ```
2. List all dictionaries

`ls`

3. Look up a value.
   `lookup SPW en_spell_iso8859_1 "apple"`

   For this example, you can omit the dictionary type and name, if you have called `with SPW en_spell_iso8859_1` before this command. This also applies to the following commands.
4. Either insert terms ...
   `insert SPW en_spell_iso8859_1 "apple"=3`
5. ... or import a text file in the dictionary's import format
   `import SPW en_spell_iso8859_1 spellingcorrection.txt`

   Type `import ?` to get the full syntax and options for this command.
6. Compile the dictionary.

   Upon completion, the compiled dictionaries can be found on the machine where the adminserver runs:

   - UNIX: `$FASTSEARCH/adminserver/webapps/adminserver/downloads/dictionaries`
   - Windows: `%FASTSEARCH%\adminserver\webapps\adminserver\downloads\dictionaries`

   The compile command returns a job number that can be used to get a summary of the job status. Use the jobstatus <id> command to get a summary that contains the job status code and a status message. A message indicates job completion.

   `compile SPW en_spell_iso8859_1`

The clientside log can be found on the machine where esp4jtool-dictman runs:

- UNIX: $FASTSEARCH/esp4j/apps/esp4jtool-dictman/log
- Windows: %FASTSEARCH%\esp4j\apps\esp4jtool-dictman\log

Search for Entries in a Dictionary

Dictman allows you to search for entries within a dictionary. This is the way to verify that a dictionary includes the correct entries.

1. Start dictman.
2. Search for any synonym keys in the synonym dictionary *en_synonyms* . "c"

```
Syntax: query <dictionary type> <name> <prefix>
query SDD en_synonyms "c"
```

   This query will return all keys in the dictionary starting with the letter *c* . If the prefix is left empty, all entries will be printed, 50 at a time.

Re-Processing Documents after a Configuration Change

The following describes the reprocessing of documents after configuration and dictionary changes on the document side.

Reprocessing of documents is required after changes in lemmatization dictionaries. For the changes to take effect, reprocessing is necessary for document side synonym dictionaries, entity dictionaries, vectorizer dictionaries and offensive content dictionaries.

Documents processed before configuration changes are made will need to be re-processed if you want these configuration changes to apply to them as well.

Documents processed after the configuration changes are made will be processed with the configuration updates.

Tip:

Re-processing may take an extended period of time depending on your configuration; make sure you allow time for documents to be completely re-processed.

If documents in FAST ESP came from sources other than the crawler, sources such as the File Traverser, database connector or content API, then these documents must be re-inserted.

This procedure applies only to crawled documents. For other pushed content, the documents must be pushed again using the same mechanism as used originally.

1. Stop the Crawler from the **System Management** tab in the Administrator interface.
2. Go to $FASTSEARCH/bin (Unix) or %FASTSEARCH%\bin (Windows).
3. Set up the required environment variables (if not already set in the command line shell):
   `source ./setupenv.sh or ./setupenv.csh`
4. Reprocess content

   To re-process a specific collection.

   ```
   ./postprocess -o -I master -R collection_name
   ```

   To re-process all collections.

   ```
   ./postprocess -o -I master -R "*"
   ```
5. Wait until the postprocess command has terminated.
6. Restart the crawler.

Editing and Importing Anti-Phrase Dictionaries

Antiphrase dictionaries contain single-term and multi-term stopwords that should be removed from queries.

Entry format

The entry format for antiphrase dictionaries is:

```
ENTRY
```

Antiphrase dictionaries are normally encoded in UTF-8 (both source and compiled dictionary). Terms in antiphrase dictionary do not have any associated information, i.e. they are a plain list of stop words and stop phrases.

Examples:

```
where can I find information on
how to
the
```

Editing and Importing Spell Checking Dictionaries

Spellcheck dictionaries - SPELLCHECK_WORDS(SPW) and SPELLCHECK_PHRASES (SPP) - contain words and phrases that should be spellchecked. Words are associated with frequency information which is used to decide which correction is selected. Generally, single-word spell checking dictionaries are used for specific languages, whereas advanced phrase spell checking dictionaries are used for queries in all languages. The latter are also used for automatic phrasing of queries.

Entry format

The entry format for spell check dictionaries of both types is:

`ENTRY.FREQUENCYINFORMATION`

In simple spell check dictionaries, ENTRY must be one single word, in advanced phrase spell checking dictionaries, it can be several words (most of the time two words).

The frequency information is a logarithmic value based on the frequency Fmax for the most frequent word in a corpus, calculated as log(Fmax/Fterm). The higher the value, the lower the frequency in the corpus.

If you choose a frequency manually when entering a word: 2 is an extremely high frequency - the word will always be preferred over other words for spell checking; 15 would be a extremely low frequency, the word will only be selected as a correction, if no term with a higher frequency is available.

Examples of entries in a simple spell check dictionary

```
and<TAB>2
antidisestablishmentarianism<TAB>14
implement<TAB>5
zombie<TAB>8
```

Examples of entries in an advanced phrase spell check dictionary

```
astrid lindgren<TAB>5
carbon copy<TAB>6
new jersey<TAB>3
zip code<TAB>3
zurich financial services<TAB>8
```

Note: Dictionaries of type SPELLCHECK_PHRASES may also contain single words.

Editing and Importing Spellcheck Exception Dictionaries

Spellcheck exception dictionaries (SPELLCHECK_EXCEPTIONS,EXC) are lists of terms that should should not be spellchecked.

Entry format

The entry format for spellcheck exception dictionaries is:

`ENTRY`

A spell check exception dictionary is a flat list of terms that should not be considered for spell checking. The compiled version of spellcheck exception dictionaries is normally encoded in the encoding the main spell check dictionary is encoded in.

Examples:

```
shakspere
```

Editing and Importing Lemmatization Dictionaries

Lemmatization dictionaries (LEMMATIZATION, LD) contain mappings of single terms to other forms of the same term.

Entry format

The entry format for lemmatization dictionaries is:

```
BASE_FORM<separator>[FORM1,FORM2,FORM3...]
```

Lemmatization dictionaries are normally encoded in UTF-8 (both source and compiled dictionary). Terms in lemmatization dictionaries contain mappings to all full forms of one common base form, and include the part of speech of the base form. Parts of speech can be N (Nouns), Adjectives (A) and Verbs (V).

The <separator> between key and value is `tab` for importing a file and = for the insert command within dictman.

Examples:

```
car<TAB>[cars,car]
walk<TAB>[walks,walking,walk,walked]
```

Editing and Importing Lemmatization Blacklist Dictionaries

Lemmatization Blacklist dictionaries (LEMMATIZATION_BLACKLIST,LB) are lists of terms that should be blocked from lemmatization.

Entry format

The entry format for lemmatization blacklist dictionaries is:

```
ENTRY
```

A lemmatization blacklist dictionary is a flat list of terms that should not be considered for lemmatization. The file must be encoded in UTF-8. It can contain single-word terms and multi-word terms. All entries should be lowercase.

Examples:

```
houses
extensible markup language
```

Editing and Importing Query Side Synonym Dictionaries

Query side synonym dictionaries (QUERY_SYNONYM_EXPANSION,SDQ) contain single-term and multi-term synonyms that serve as suggestions and/or modifications for user queries.

Entry format

The entry format for query side synonym dictionaries in Wirth Syntax Notation (WSN) is as follows (note that the square brackets in the following denote square brackets, not optional items. Optional items are shown in parentheses):

```
PHRASE<SEPARATOR>[MODIFICATIONS,SUGGESTIONS]
MODIFICATIONS = [REWRITEFLAG,TERMLISTMOD]
TERMLISTMOD = TERMMOD,(TERMLISTMOD)
TERMMOD = [PHRASE,WEIGHT,SYMMETRIC]
SYMMETRIC = BOOL
SUGGESTIONS = [SYNSET,(SUGGESTIONS)]
SYNSET = [TERMLISTSUGG]
TERMLISTSUGG = TERMSUGG,(TERMLISTSUGG)
TERMSUGG = [PHRASE,WEIGHT,SYMMETRIC,UNUSED]
PHRASE = WORD (PHRASE)
WORD = word_character+
BOOL = true|false
WEIGHT = integer
```

Some explanations:

- MODIFICATIONS contains terms that are used for changing the query.
- SUGGESTIONS contains terms that are only issued as suggestions.
- REWRITEFLAG indicates whether this terms should rewrite the query by omitting the original term (replace) or just add it as an alternative to the original term
- SYMMETRIC indicates, whether the term pair should also be added in the opposite direction.
- WEIGHT is the weight of the term: 100 is the default value, 1 is ignored. Can be set to values between 2 and 1000 to indicate the weight of the replacement term. Ignored for suggestions.
- SUGGESTIONS are grouped into synsets - this grouping will be reflected in the feedback.
- The <separator> between key and value is `tab` for importing a file and = for the insert command within dictman.

Synonym dictionaries are always encoded in UTF-8 (both source and compiled dictionary).

Example for setting up modification terms:

```
    space flight<TAB>[[false,[spaceflight,1,true],[space
exploration,1,false]],[]]
    spaceflight<TAB>[[true,[space flights,1,false],[space
flights,1,false]],[]]
```

In the first example, for the query *space flight* the variants *spaceflight* and *space exploration* will be added. Because of the flag 'true' set for *spaceflight*, *spaceflight* will also be expanded to *space flight*

In the second example, the query will be rewritten, i.e. *spaceflight* will be removed from the query.

Example for setting up suggestion terms:

```
    brain<TAB>[[],[[[wit,1,false,false],[mental
capacity,1,false,false]],[[mastermind,1,false,false],[genius,1,false,false]]]]
```

This example sets up two synonym groups as suggestions for the word *brain*

Editing or Importing a Document-Side Synonym Expansion Dictionary

Dictionaries for document-side synonyms (DOCPROC_SYNONYM_EXPANSION, SDD) contain synonyms for expansion on the document side, formatted as key-value pairs: the key denotes the term to be extended, and the value contains a list of expansion terms.

There are three methods for adding terms to a document side synonym expansion dictionary:

- Adding terms via the command line in dictman.
- Importing a TEXT file using dictman.
- Importing an XML file using dictman.

Entry format

The entry format for document side synonym dictionaries is:

```
TERM<separator>[SYN1,SYN2...]
```

Synonym dictionaries are normally encoded in UTF-8 (both source and compiled dictionary).

The <separator> between key and value is `tab` for importing a file and = for the insert command within dictman.

Examples:

```
car<TAB>[automobile,auto]
sailing boat<TAB>[sail boat,sailboat]
```

Import Content from an XML File into a Document-Side Synonym Dictionary

Dictman allows you to import content from an xml file into a document side synonym dictionary.

This command imports the file mysynonyms.xml, which is UTF-8 encoded, into the dictionary.

The format of keys and values is the same as when inserting terms via the command line, but they must not be enclosed in quotes.

1. Start esp4jtool-dictman.
2. Import entries from the xml file.

   The xml file should look like this:

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <!--
 This is an example file that shows how to create an
     XML input file for qt_synonym in ESP 5.X

  -->
- <!--
 Paths are not supported here, but you can supply a URI
  <!DOCTYPE thesaurus SYSTEM "thesaurus_format.dtd">

  -->
- <thesaurus>
- <!--
 REWRITE TERMS - lead to query modification where all synonyms
```

```
                       are added to the query, joined with original term with ANY
       - automatic rewrite,
       - unidirectional.
       You use this for all terms that you want automatically rewritten,
       synonyms, variations etc...
       The weight will be used for all spellVar terms (100 or 1 is default, all other
 weights
       interpreted based on default weight 100)

  -->
- <term lang="en" weight="1" id="tr1">
  <label>organization</label>
  <spellVar>organisation</spellVar>
  <spellVar>establishment</spellVar>
  </term>
- <term lang="en" weight="80" id="tr1">
  <label>sailing boat</label>
  <spellVar>sailboat</spellVar>
  <spellVar>sailship</spellVar>
  </term>
- <!--
 SUGGESTIONS
     All following is emitted as suggestions only (no query rewrite)

  -->
- <!--  How to define a set of suggestion terms:
  -->
- <!--  STEP 1: define the terms
  -->
- <term lang="en" weight="1" id="t1">
  <label>computer</label>
  </term>
- <term lang="en" weight="1" id="t2">
  <label>computing machine</label>
  </term>
- <term lang="en" weight="1" id="t3">
  <label>data processor</label>
  </term>
- <!--
 STEP 2: define the synonym set.
      Each synonym is mapped to all the others.
      The terms are coded as references to terms
      defined in step 1

  -->
- <concept id="c1">
- <synSet>
  <termRef resource="#t1" />
  <termRef resource="#t2" />
  <termRef resource="#t3" />
  </synSet>
  </concept>
  </thesaurus>
```

```
Syntax: import <dictionary type> <name> <path to xml file>  [options]
import SDD en_synonyms /path/to/file/synonym_updates.xml
```

3. Compile the dictionary.

Editing and Importing Offensive Content Filter Dictionaries

Offensive Content Dictionaries (OFFENSIVE_CONTENT_DETECTION,OC) are lists of terms with properties that should be used for blocking documents from processing because of unwanted content.

Entry format

The entry format for offensive content dictionaries is:

`TERM<separator>=[[LANGUAGE,SCORE,AMBIGUITY,TRANSLATION]]`

`LANGUAGE` is the ISO-639 language code of the language of the offensive term.

`SCORE` is an integer value between 1 and 9, the higher, the more blocking force is attributed to the term

`AMBIGUITY` is 'a' or 't'. 'a' means, that this term represents an offensive term under all circumstances ('alone'). It is not ambiguous and does not appear in non-offensive contexts. 't' means that the term might also appear under non-offensive contexts and only counts if other offensive terms are found.

`TRANSLATION` An optional translation of the term, solely used for maintainability, or "".

Examples:

```
sex<TAB>[[en,5,t,""]]]
putains nues<TAB>[[fr,5,a,"nude whores"]]]
```

Editing and Importing Entity Dictionaries

Entity dictionaries (ENTITY_EXTRACTION, ED) are lists of entities that should should be extracted (whitelists) or that should be blocked from extraction (blacklists).

Entry format

The entry format for entity dictionaries is:

`ENTRY`

or (whitelists only):

`ENTRY<separator>CANONICAL_FORM`

Entity dictionaries are normally encoded in UTF-8 (both source and compiled dictionary). Entity dictionaries are case sensitive. Terms in entity dictionaries do not normally have any associated information, i.e. they are a plain list of words and phrases. However, a canonical form can optionally be added to whitelist entries.

Examples:

```
William Shakespeare
Willy Shakespeare<TAB>William Shakespeare
```

Index